Carl Ulrich Wassermann

Apple IIc

Handbuch für Anwender und Programmierer

Hüthig

Carl Ulrich Wassermann · Apple IIc

Carl Ulrich Wassermann

Apple IIc

Handbuch für Anwender und Programmierer

Dr. Alfred Hüthig Verlag Heidelberg

Diejenigen Bezeichnungen von im Buch genannten Erzeugnissen, die zugleich eingetragene Warenzeichen sind, wurden nicht besonders kenntlich gemacht. Es kann also aus dem Fehlen der Markierung ® nicht geschlossen werden, daß die Bezeichnung ein freier Warenname ist. Ebensowenig ist zu entnehmen, ob Patente oder Gebrauchsmusterschutz vorliegt.

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Wassermann, Carl Ulrich:
Apple IIc / Carl Ulrich Wassermann. – Heidelberg : Hüthig, 1986.
ISBN 3-7785-1157-2

© 1986 Dr. Alfred Hüthig Verlag GmbH Heidelberg
Printed in Germany
Satz und Druck: Laub GmbH & Co., 6857 Elztal-Dallau

Vorwort

Das Buch wendet sich an Benutzer eines Apple //c Computers, denen die herstellerseitig mitgelieferte Dokumentation nicht ausreichend auf wichtige Details eingeht. Dieses Buch soll Ihnen helfen, sich mit den spezifischen Möglichkeiten dieses Computers, insbesondere bezüglich Programmierung vertraut zu machen.

Die ersten Kapitel beschreiben leicht verständlich die Maschine und ihre Bedienung. Nützliche Programme wie direkter Zugriff auf Disketten-Filenamen oder zur Datumseingabe eröffnen zusätzliche Möglichkeiten des Computers. Die wesentlichen Unterschiede gegenüber den Parallelmodellen Apple // und Apple //e werden besonders hervorgehoben, weil sie gerade für den ungeübten Benutzer wichtig sind.

Die Programmierung in der Sprache BASIC ist in Kapitel drei ausführlich dargestellt. Die detaillierte Beschreibung von einzelnen Befehlen bis zu Programmsystemen erlaubt Ihnen die Formulierung leistungsfähiger und sicherer Programme. Die Abschnitte Strukturierung und Programmierhilfen ermöglichen den effizienten Einsatz der Maschine.
Der beste Effekt wird erzielt, wenn der Benutzer die dargestellten Programme auf einem Computer sofort testen und variieren kann.

Kapitel vier gibt Ihnen einen Einblick in die Programmiersprachen

- Assembler
- Pascal
- FORTH

und weitere, für den Apple //c erhältliche Sprachen. In kurzen Beispielen werden Unterschiede und Eigenschaften der Sprachen verdeutlicht. Detailliert ist beschrieben, wie Assemblerlistings eingegeben und gespeichert und eigene Maschinenprogramme formuliert werden.

Der Anschluß von Druckern und anderen Geräten ist in Kapitel fünf dargestellt. Ausführlich werden die Eigenschaften der Schnittstellen, vor allem die interaktive oder programmgesteuerte Anpassung der seriellen Ports an Peripheriegeräte, beschrieben.

Eine Darstellung wichtiger Bücher, Zeitschriften und Computerclubs erlaubt Ihnen gezielten Zugriff auf weitere Informationen.

Im Anhang sind nochmals alle DOS 3.3-, ProDOS- und Applesoft-Befehle alphabetisch geordnet zusammengestellt und beschrieben. Wichtige, vor allem Apple //c spezifische Speicheradressen ermöglichen die volle Nutzung der Maschine. Alle wichtigen Daten des Computers sind in Tabellen zusammengefaßt; sie sind für den Programmierer eine konzentrierte, wichtige Grundlage. Besonderer Wert wird dabei auf die Unterschiede des Apple //c zu den Parallelmodellen gelegt.

Inhalt

Vorwort 5

1. Der Apple //c 15
1.1 Der transportable Computer 15
1.2 Tastatur 16
1.3 Bildschirm 18
1.4 Stromversorgung 20
1.5 Diskettenlaufwerke 20
1.6 Anschlüsse 21
1.7 Die Betriebssysteme 22
1.8 Unterschiede gegenüber dem Apple //e 23

2. Die Bedienung des Apple //c 25
2.1 Inbetriebnahme 25
2.2 Inhaltsverzeichnis der Disketten 27
2.3 Betriebssysteme 29
2.3.1 ProDOS 30
2.3.1.1 Datum und Uhrzeit unter ProDOS 31
2.3.1.2 Filestruktur 31
2.3.2 DOS 3.3 34
2.3.3 UCSD Pascal 34
2.4 Disketten formatieren 35
2.4.1 Formatieren unter DOS 3.3 36
2.4.2 Formatieren unter ProDOS 37
2.5 Handhabung der Disketten 38
2.6 Pflege des Computers 39

3. Applesoft BASIC 41
3.1 Einzelanweisungen
(PRINT, ?) 42

3.2 Programmzeilen (LIST, RUN) 44
3.3 Programme auf Disketten speichern, laden, auflisten, ändern (CAT, CATALOG, SAVE, LOAD, NEW, END, DELETE, LOCK, UNLOCK, VERIFY, RENAME, HOME) 46
3.4 Der Editor (ESC, Cursortasten, DEL) 50
3.5 Variable, Daten, Texte, Zahlen (DIM, FRE(0)) 54
3.5.1 Numerische Variablen 56
3.5.2 Textvariablen 57
3.5.3 Matrizen (Arrays, Felder) 58
3.5.4 Freier Speicherplatz, FRE (0) 60
3.6 Rechenoperationen (+, -, *, /, ↑, =, ungleich, NOT, AND, OR) 61
3.6.1 Vergleichende Ausdrücke 63
3.6.2 Boolsche Operationen 64
3.7 Programmverzweigungen (GOTO, GOSUB, RETURN, ON..GOTO, ON..GOSUB, POP, IF..THEN, FOR..NEXT) 66
3.7.1 Einfache Sprünge, GOTO 67
3.7.2 Unterprogramme, GOSUB ... RETURN 69
3.7.3 Programmschleifen, FOR ... NEXT 73
3.7.4 Bedingte Verzweigungen, IF ... THEN 77
3.8 Eingabe und Ausgabeprogramme (INPUT, GET, VAL, PRINT, SPC, HOME, TAB, VTAB, HTAB, INVERSE, NORMAL, FLASH, SPEED=, CHR$, ASC, WAIT) 78
3.8.1 Eingabebefehle 49
3.8.1.1 Der INPUT Befehl 79
3.8.1.2 Der GET Befehl 81
3.8.2 Ausgabebefehle 82
3.8.2.1 Der PRINT Befehl 82
3.8.2.2 Der Befehl SPC( ) 84
3.8.2.3 Die Befehle TAB, HTAB, VTAB 85
3.8.2.4 Die Befehle INVERSE, FLASH, NORMAL, SPEED= 87
3.8.2.5 Die Befehle CHR$, ASC( ) 89
3.8.3 Gestaltung von Eingabe- und Ausgabeprogrammen 90

3.8.3.1 Eingabeprogramme 90
3.8.3.2 Ausgabeprogramme 91
3.9 Daten im Programmtext
(DATA, READ, RESTORE) 92
3.10 Funktionen
(SGN, ABS, INT, SQR, EXP, LOG, RND,
SIN, COS, TAN, ATN, ASC, LEN, VAL,
CHR$, STR$, LEFT$, RIGHT$, MID$) 95
3.10.1 Numerische Funktionen 96
3.10.2 Funktionen definieren, DEF FN 97
3.10.3 Textfunktionen 98
3.11 Anweisungen an das Betriebssystem
(CHR$(4), CHAIN, EXEC, FLUSH, FRE, FP, IN#,
INT, MAXFILES, MON, NOMON, PR#,
INIT, PREFIX, RESTORE, STORE, CREATE, -) 98
3.12 Der direkte Zugriff zum Speicher
(PEEK, POKE, CALL, LOMEM:, HIMEM:) 105
3.12.1 Speicherinhalte lesen, PEEK 106
3.12.2 Speicherinhalte ändern, POKE 107
3.12.3 Speicheraufteilung 109
3.12.4 Hexadezimale Adressen 110
3.13 Zugriff auf Disketten
(OPEN, APPEND, CLOSE, READ, WRITE,
POSITION) 111
3.13.1 Das 'OPEN' Problem 113
3.13.2 Files schließen, CLOSE und MAXFILES 115
3.13.2.1 Zwischenaktivierung des Basic Interpreters 116
3.13.3 Sequentielle Textfiles 117
3.13.3.1 Records, Felder, Bytes 118
3.13.3.2 Daten einlesen mit GET 122
3.13.3.3 END OF DATA Fehler 122
3.13.3.4 Daten anfügen, APPEND 123
3.13.3.5 Zugriff auf einen bestimmten Eintrag, POSITION 124
3.13.4 RANDOM ACCESS Files (mit direktem Datenzugriff) 124
3.13.4.1 Blockweise Datenspeicherung 126
3.13.4.2 Feld- und Byteweiser Datenzugriff 127
3.13.5 Abkürzungen 128
3.13.6 Filezugriff mit ProDOS 128
3.13.6.1 Filepuffer leeren, FLUSH 128
3.13.6.2 Feldweiser Datenzugriff 129

3.13.6.3 Angabe des Filetyps 129
3.14 Grafik
(GR, HGR, HGR2, COLOR=, HCOLOR=, HLIN, HPLOT, DRAW, SCALE=, ROT=, XDRAW, SCRN, TEXT) 130
3.14.1 Niedrigauflösende Grafik, GR 130
3.14.1.1 40 x 40 (40 x 48) Bildpunkte 130
3.14.1.2 80 x 40 (80 x 48) Bildpunkte 133
3.14.2 Hochauflösende Grafik 133
3.14.2.1 Umschalten der Grafikseiten 136
3.14.2.2 Linienfolgen 136
3.14.2.3 Grafik und Speicherplatz 141
3.14.2.4 Bewegte Bilder 142
3.14.3 Grafik und Diskette 144
3.14.3.1 Grafik auf Diskette speichern 144
3.14.3.2 Grafik von Diskette einlesen 146
3.15 Programmierung der Maus 146
3.15.1 Programmierung in BASIC 146
3.15.1.1 Grafikprogramme mit der Maus 148
3.15.1.2 Hochauflösende Grafik auf Diskette speichern 151
3.15.1.3 Grafiken invertieren 152
3.15.2 Firmwareroutinen für die Maus 152
3.16 Strukturierte Programmierung 155
3.16.1 Übergabeparameter 155
3.16.2 Unterprogramme 156
3.16.2.1 Eingabeprogramm 156
3.16.2.2 Ausgabeprogramm 158
3.16.2.3 Sequentielles Zugriff auf Diskfiles 158
3.16.2.4 Text-Hardcopy auf Drucker 159
3.16.3 Programmverkettung, Chaining 159
3.17 Symbolische Sprungadressen, strukturiertes Basic 162
3.18 Kompakte Programme 166
3.19 Schnelle Programme 167
3.20 Basic Compiler 169
3.21 Programmierhilfen
(STOP, CONT, TRACE, NOTRACE, MON, NOMON, SPEED= Ctrl-S, Ctrl-C) 170
3.21.1 Vorgehensplan für die Programmentwicklung 170
3.21.2 Fehlersuche, Fehlerabfangung 171
3.21.2.1 Programmablauf, TRACE 171
3.21.2.2 Programm unterbrechen, STOP, CONT 172

3.21.2.3 Hilfsroutinen 173
3.21.2.4 Filezugriff kontrollieren 174
3.21.2.5 Programme verzögern, anhalten 174
3.21.2.6 Fehlerabfangung 175
3.21.3 Hilfsprogramme 176
3.21.4 Programme mit Textverarbeitung ändern 178
3.22 Verwendung der RAM - Disk 181
3.22.1 Zwischenspeicher für Programmteile 182
3.22.2 Zwischenspeicher für Programmdaten 184
3.22.3 Zwischenspeicher für Hires Grafiken 185

4. Andere Programmiersprachen 186
4.1 Ein Überblick 186
4.2 Pascal 188
4.2.1 Pascal Programme 189
4.3 Forth 191
4.3.1 Notation 195
4.3.2 Der Stack 196
4.3.3 Programmschleifen und Vergleiche 198
4.3.4 Zahlensysteme 199
4.3.5 Screens und Editor 200
4.4 Assembler 201
4.4.1 Die Eingabe kurzer Maschinenprogramme 202
4.4.1.1 Eingabe über ein Basic Programm 203
4.4.1.2 Direkte Eingabe von Maschinenprogrammen 203
4.4.1.3 Maschinenprogramme auf Diskette speichern 206
4.4.2 Assemblerprogramme 207
4.4.2.1 Der Miniassembler 208
4.4.2.2 Andere Assembler 210
4.4.2.3 Assemblersyntax 211
4.4.2.4 Lade- und Speicherbefehle 213
4.4.2.5 Befehle für arithmetische Operationen 214
4.4.2.6 Stackoperationen 216
4.4.2.7 Logische Operationen 216
4.4.2.8 Vergleichsoperationen 217
4.4.2.9 Verzweigungsbefehle 217
4.4.2.10 Der Interruptbefehl 218
4.5 Monitorbefehle 219
4.5.1 Speicherinhalte listen und disassemblieren 219
4.5.2 Speicherinhalte abfragen 219

4.5.3 Speicherinhalte ändern 220
4.5.4 Speicherinhalte kopieren 220
4.5.5 Sprünge im Monitor 222
4.5.6 Ausgabe über Drucker 222
4.5.7 Registerinhalte des Prozessors ändern 223
4.5.8 Videoeffekte 224
4.5.9 Hexadezimale Arithmetik 224
4.5.10 Verlassen des Monitors 224

5. Anschluss externer Geräte 226
5.1 Die Slots/Ports des Apple //c 226
5.2 Serielle Ports des Apple //c 229
5.2.1 Serielle Ports konfigurieren 230
5.2.1.1 Baudrate 230
5.2.1.2 Datenformat 231
5.2.1.3 Zeilenbreite 233
5.2.1.4 Echo 233
5.2.1.5 Zeilenvorschub nach Carriage RETURN (CR) 233
5.2.1.6 Standardwerte setzen 234
5.2.1.7 Break 234
5.2.1.8 ZAP 234
5.2.1.9 Andere Kontrollzeichen 235
5.2.1.10 Spezifische Port 2 Befehle 235

6. Weiterführende Informationen 237
6.1 Bücherverzeichnis 238
6.2 Zeitschriften 241

Anhang
Anhang 1 Applesoft- und DOS-Anweisungen (alphabetisch) 242
Anhang 2 Zusammenfassung der Betriebssystemanweisungen 257
Anhang 3 Interne Unterschiede Apple //c zum Apple //e 261
Anhang 4 Fehlermeldungen 265
Anhang 5 Reservierte Worte 271
Anhang 6 Speicheraufteilung 273
Anhang 7 ASCII Code 277
Anhang 8 Pinbelegung der Anschlüsse 281
Anhang 9 Maschinenprogramme und Adressen 284

Anhang 10 Hardwarepage Adressen ($C000 - $C100) 293
Anhang 11 Maschinenbefehle des 6502 und des 65C02 Prozessors 298

Stichwortverzeichnis 321

Kapitel 1

Der Apple //c Computer

1.1 Der portable Computer

Der Apple //c wurde als tragbarer (portabler) Computer entwickelt und wird als solcher auch von Apple vermarktet. Obwohl er wesentlich handlicher (3,4 kg) ist als die meisten anderen „tragbaren Computer", ist der Benutzer wegen der Stromversorgung doch stark eingeschränkt.

Für den Apple //c werden verschiedene Bildschirme (Monitore) angeboten. Wirklich tragbar davon ist aber lediglich der flache LCD-Schirm, der keine eigene Stromversorgung benötigt.

Betreibt man den Computer an zwei Orten, zum Beispiel nur zu Hause und im Büro, so ist die Anschaffung eines zweiten Monitors und Netzteils ratsam, die ja nur einen Bruchteil des Apple //c kosten.

Der Apple //c ist der äußerlich kleinste Computer von Apple, allerdings nicht der am wenigsten leistungsfähige.

Der Apple //c verfügt serienmäßig über 128 kB Speicher-RAM, die nicht mehr erweitert werden können, und 16 kB ROM, von dem 15,75 kB genutzt werden. Der ROM Speicher enthält im Wesentlichen die Sprache BASIC (Applesoft) und einige für den Computer notwendigen Informationen, zum Beispiel, daß er nach dem Einschalten zunächst im eingebauten Laufwerk eine Diskette suchen soll.

Die Informationen im ROM Speicher können nicht verändert werden.

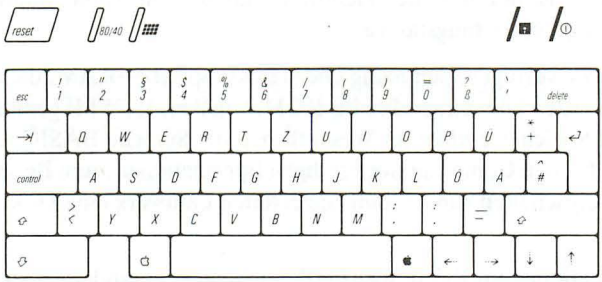
Im RAM Speicher behält der Computer, solange er nicht ausgeschaltet wird, das Benutzerprogramm, welches von der Diskette gelesen oder über die Tastatur eingegeben wird, sowie die beim Ausführen des Programms notwendigen Daten.

Der Apple //c wurde aus dem Apple //e entwickelt. Er ist aber nicht dessen Nachfolgemodell, sondern mehr eine tragbare Version. Der Apple //c ist ähnlich aufgebaut wie der Apple //e, weist aber einige wesentliche Unterschiede auf. Diese Unterschiede sind am Ende des Kapitels und im Anhang beschrieben.

Das Computergehäuse enthält eine einzige Platine, auf der der Mikroprozessor 65C02 (der eigentliche „Rechner") und die weiteren Chips untergebracht sind, ein eingebautes 5 1/4 Zoll Diskettenlaufwerk, Anschlüsse für Strom, Drucker usw. auf der Rückseite und eine Tastatur. Auf diese Bestandteile wird in den folgenden Abschnitten eingegangen.

1.2 Die Tastatur

Der Apple //c hat eine Standardtastatur, die in Größe und Buchstabenaufteilung einer normalen Schreibmaschinentastatur entspricht. Da die Aufteilung der Buchstaben in jedem Land unterschiedlich ist, kann durch den rechten Schalter oberhalb der Tastatur zwischen zwei verschiedenen Aufteilungen gewählt werden. Die Aufteilung hängt davon ab, in welchem Land Sie den Computer gekauft haben. Im Allgemeinen können Sie dann zwischen der jeweiligen Landestastatur und der US-Tastatur umschalten.
Die US-Tastatur ist vor allem beim Programmieren von Vorteil, da alle Programmiersprachen für US-amerikanische Tastaturaufteilung ausgelegt sind.



Die Tastatur des Apple //c hat gegenüber einer Schreibmaschine einige Tasten zusätzlich, die in verschiedenen Programmen Anwendung finden:

CAPS LOCK, OPEN APPLE, SOLID APPLE sowie die vier Cursortasten LEFT ARROW, RIGHT ARROW, UP ARROW, DOWN ARROW auf der untersten Reihe, CONTROL und ESC (Escape) links, sowie DELETE rechts oben.

Die Taste CAPS LOCK (Feststelltaste) sperrt die Kleinbuchstaben, OPEN APPLE und SOLID APPLE haben je nach ausgeführtem Programm, meist in Verbindung mit anderen Tasten, unterschiedliche Bedeutung. Manchmal sind sie auch ohne Funktion.
Die vier Cursortasten werden von manchen Programmen gebraucht. Sie werden ebenso wie die ESC-Taste beim Programmieren in BASIC verwendet.
CONTROL für sich allein hat keine Bedeutung, ist aber beim Programmieren und in manchen Programmen zusammen mit anderen Tasten sehr wichtig, z.B. CONTROL-C (gleichzeitiges Drücken von CONTROL und C), das bei sehr vielen Programmen zum Abbruch führt.
DELETE löscht in vielen Programmen eingegebene Zeichen (z.B. Textverarbeitung), nicht aber bei der Programmierung in Basic.
Oberhalb der Tastatur links befinden sich außerdem RESET (führt bei fast allen Programmen zusammen mit CONTROL zum Abbruch desselben) und der Schalter 80/40, mit dem beim Programmieren zwischen der Darstellung mit 40 und 80 Zeichen pro Zeile gewählt werden kann.
Die RETURN-Taste rechts unter DELETE hat mehrere Bedeutungen je nach Programm. Im Allgemeinen schließt sie eine Eingabe ab, und der Computer fährt in seinem Programm fort.

Die Tastatur ist der wichtigste Mechanismus, um den Computer direkt mit Informationen zu versorgen. Daneben existieren weitere Möglichkeiten zur Dateneingabe:

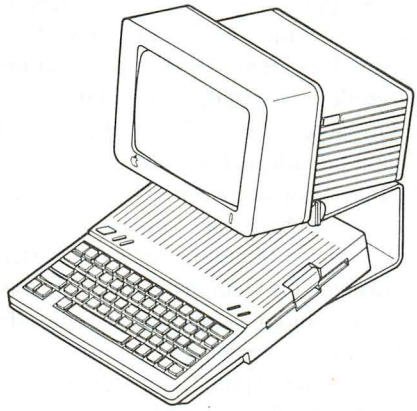
Paddles und Joysticks dienen vor allem bei Spielprogrammen zur Steuerung. Die Maus wird bisher nur in wenigen Programmen eingesetzt z.B. zum Zeichnen (siehe Kapitel 3). Sie kann auch bei eigenen Programmen verwendet werden. Paddles, Joysticks und die Maus werden auf der Rückseite des Computers angeschlossen.
Spracheingabe ist bisher nicht üblich, dagegen werden für den Apple//c Geräte für die Sprachausgabe angeboten.

1.3 Der Bildschirm

Der Apple//c kann mit einem handelsüblichen Fernseher oder mit Monitor betrieben werden.
Wird ein Fernseher benutzt, so sollte nur die 40 Zeichen Darstellung verwendet werden, da die Bildqualität bei der Darstellung von 80 Zeichen pro Zeile sehr schlecht ist. Fernsehgeräte werden über den mitgelieferten PAL-Modulator angeschlossen.
Wird der Apple //c nicht nur sporadisch genutzt, so ist ein Monitor empfehlenswert, der an den zweipoligen Monitorausgang auf der Rückseite des Computers angeschlossen wird. Damit werden nicht nur die Augen geschont, sondern mit einem flimmerfreien Monitor wird eine wesentlich weniger ermüdende Arbeit am Computer möglich. Besonders wichtig ist aber der dann mögliche Einsatz der 80-Zeichen Darstellung. Viele Anwenderprogramme verwenden die 80-Zeichen Darstellung (vor allem Textverarbeitung). Diese Darstellung ist übersichtlicher, da auf dem Bildschirm die gesamte Breite eines A4 Blattes und damit die doppelte Informationsmenge dargestellt werden kann.
Vertikal werden immer 24 Zeilen angezeigt. Die Darstellung von mehr als 24 Zeilen auf dem Bildschirm ist beim Apple //c wie bei den meisten anderen Computern nicht vorgesehen.

Es ist nicht notwendig, Apple Monitore zu verwenden. Monitore anderen Fabrikats haben in der Regel gleiche Bildqualität, sind aber oft billiger. Die Größe des Bildschirms und die Farbe spielen dabei keine große Rolle, lediglich der Anschluß muß zum Stecker des Kabels passen.

Diese Monitore dürfen jedoch in keinem Fall direkt auf den Computer gestellt werden, weil das Gehäuse des Apple //c nicht dafür geeignet ist. Der Computer kann durch die behinderte Lüftung zu heiß werden.
Da das im Apple //c eingebaute Laufwerk nicht ausreichend vor äußeren Magnetfeldern geschützt ist, muß der verwendete Fernseher oder Monitor einen



gewissen Abstand zur eingebauten Diskettenstation haben (ca. 40 cm). Der Original Apple //c Monitor ist abgeschirmt und kann deshalb direkt oberhalb des Laufwerks auf einem Ständer betrieben werden.

Der flache LCD-Bildschirm braucht keine eigene Stromversorgung, er wird vom Apple //c aus versorgt. Der Kontrast dieses Monitors kann nicht so stark eingestellt werden wie der von gebräuchlichen Monitoren. Deshalb ist die Lesbarkeit bei starker Bestrahlung, z.B. Sonnenlicht eingeschränkt.
Farbige Monitore sind nur in Verbindung mit farbiger Grafik sinnvoll. Praktisch alle nichtgrafischen Programme arbeiten einfarbig. Farbige RGB-Monitore sind in der Regel mehr als doppelt so teuer wie gleichwertige Schwarzweiß-Monitore und haben bei schwarzweißer Darstellung meist schlechtere Bildqualität.

Der Monitor wird zweckmäßigerweise so aufgestellt, daß sich möglichst wenig Reflexionen auf dem Schirm ergeben; Fenster oder Lampen hinter dem Benutzer, deren Licht sich im Monitor spiegelt, verschlechtern den Bildkontrast und führen zu schneller Ermüdung der Augen.

Der Apple //c hat vier Darstellungsarten:
Text (beim Einschalten des Computers aktiv) sowie niedrigauflösende Grafik in maximal 16 Farben, hochauflösende Grafik und doppelt-hochauflösende Grafik in jeweils maximal 6 Farben, die natürlich nur bei RGB-Farbmonitoren sichtbar sind. Bei einfarbigen Monitoren entstehen unterschiedliche Schattierungen.

1.4 Stromversorgung

Die Stromversorgung des Apple //c ist vom Computergehäuse getrennt. Der mit dem Computer gelieferte Netztrafo (6 x 7 x 13 cm) wird auf der Rückseite des Computers angeschlossen. Die Trennung von Computer und Stromversorgung erlaubt dem Benutzer größere Flexibilität beim Transport und bei der Aufstellung des Computers. Offenbar ist dieses Konzept auch im Hinblick auf die zu einem späteren Zeitpunkt erhältliche Batterieversorgung gewählt worden. Der Apple //c benötigt die Betriebsspannungen +/- 5 Volt und +/- 12 Volt.
Der Batteriebetrieb wird jedoch erst bei Verwendung des flachen LCD-Bildschirms sinnvoll, da die jetzigen Monitore einen für Batteriebetrieb zu hohen Strombedarf haben, und damit die Gebrauchsdauer des Computers mit Batterie stark reduzieren. Die Auswahl an Monitoren, die mit Gleichstrom niedriger Spannung (z.B. 12 V) betrieben werden können, ist klein.
Wärmeprobleme, die bei älteren Apple Computern unter ungünstigen Bedingungen vorgekommen sind, werden mit der Verwendung eines externen Netzteils eliminiert. Trotzdem ist beim Betrieb des Computers darauf zu achten, daß die Lüftungsschlitze auf Unter- und Oberseite nicht abgedeckt werden.

1.5 Die Diskettenlaufwerke

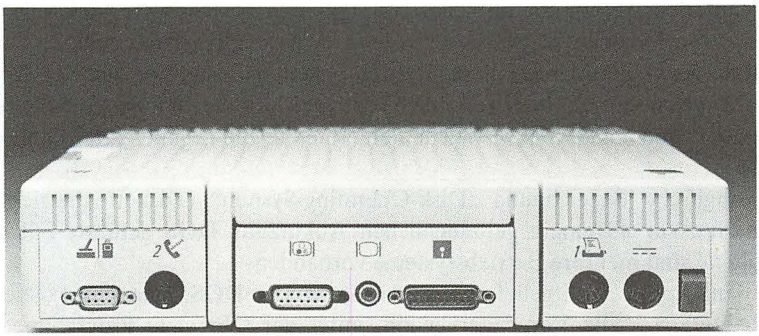
Der Apple //c hat ein eingebautes Laufwerk für 5 1/4 Zoll Disketten auf der rechten Seite. Dazu kann maximal noch ein weiteres externes Laufwerk an der Rückseite des Computers angeschlossen werden. Die Speicherkapazität der Disketten ist auf etwa 143.000 Byte (140 kB) begrenzt. Davon stehen dem Benutzer ca. 135 kB (je nach Betriebssystem etwas mehr oder weniger) zur Verfügung. Dies ist verglichen mit der Kapazität der Disketten anderer Computerhersteller wenig, hat aber den Vorteil der größeren Datensicherheit (die Diskette ist weniger empfindlich).
Die Kapazität der Disketten reicht für den allergrößten Teil der Anwendungen aus. Die erwähnten 135 kB pro Diskette entsprechen etwa 35 vollgeschriebenen Schreibmaschinenseiten Text.
Die Kapazität der Disketten wird nicht durch die Diskette selbst bestimmt, sondern durch das Aufzeichnungsverfahren des Apple //c. Dieser schreibt die Informationen auf 35 vom Auge nicht sichtbare konzentrische Ringe (Spuren oder Tracks), die in jeweils 16 Sektoren eingeteilt sind (zu je 256 Byte).

Disketten, die anders formatiert (aufgeteilt) sind, kann der Apple //c nicht lesen. Die Disketten können auch auf der Rückseite beschrieben werden; siehe hierzu Kapitel 2.
Die eingebaute Diskettenstation ist wie das von Apple erhältliche externe Laufwerk wartungsfrei. Sollten dennoch einmal Störungen beim Schreiben oder Lesen auf Diskette auftreten, so liegt das mehrheitlich an der Diskette selbst, oder selten auch an der Drehzahl des Laufwerks, die der Händler dann einstellen kann.
Diskettenlaufwerke für den Apple //e werden auch von anderen Herstellern als Apple abgeboten, können aber an den Apple //c nicht angeschlossen werden, da die Stecker und die Verdrahtung unterschiedlich sind. Beim zweiten Laufwerk muß also entweder ein Original Apple //c Laufwerk verwendet werden oder ein anderes, welches ausdrücklich an den Apple //c angeschlossen werden kann. Dank der Tragbarkeit des Apple //c ist es ja kein Problem, ihn zum Händler mitzunehmen und das gewünschte externe Laufwerk auszuprobieren.

Die Behandlung der Disketten ist in Kapitel 2 beschrieben.

1.6 Die Anschlüsse

Da der Apple //c im Gegensatz zum Apple //e nicht erweitert werden kann, sind praktisch alle notwendigen Anschlüsse schon serienmäßig eingebaut.



Die Anschlüsse auf der Rückseite des Apple //c sind (bei Draufsicht) von links nach rechts:

9 poliger Anschluß für die Maus, Paddle oder Joystick.
5 poliger serieller DIN Ausgang für ein Modem, Koppler, Drucker oder ähnliches.
15 poliger Anschluß für einen Farbmonitor, flachen Bildschirm oder Fernseher.
2 poliger Anschluß für einen einfarbigen Monitor.
19 poliger Anschluß für ein zweites Diskettenlaufwerk.
5 poliger serieller DIN Ausgang für ein Modem, Koppler, Drucker oder ähnliches.
7 poliger Anschluß für die Stromversorgung.

Zusatzgeräte wie Drucker, Maus etc., die von der Firma Apple für den Apple //c angeboten werden, geben keine Probleme beim Anschließen; in den mitgelieferten Unterlagen ist das Vorgehen gut dokumentiert. Bei der Verwendung anderer Produkte kann es in manchen Fällen Schwierigkeiten geben. Näheres dazu steht in Kapitel 5.

Auf der linken Seite des Apple //c ist neben dem Lautstärkeregler ein 2 poliger Ausgang für einen Lautsprecher vorhanden. Wird ein externer Lautsprecher angeschlossen, so wird der eingebaute auf der Unterseite vorne in der Mitte stillgelegt.

1.7 Die Betriebssysteme

Das mit dem Computer mitgelieferte (teils fest gespeicherte und teils auf Diskette vorhandene) Betriebssystem dient im Wesentlichen der internen Organisation des Apple. Durch dieses Betriebssystem ist festgelegt, wie der Apple auf Disketten schreibt oder von ihnen liest, welche Eingaben des Benutzers er akzeptiert und ausführt bzw. welche Fehlermeldungen gegebenenfalls notwendig sind.
Die englische Bezeichnung „Disk-Operating-System" (Diskettenbetriebssystem) hat zur allgemein gebräuchlichen Abkürzung DOS geführt. Für den Apple //c sind mehrere Betriebssysteme vorhanden.
Mit dem Apple //c mitgeliefert wird das System ProDOS. Daneben existieren noch DOS 3.3, die Vorgängerversion, unter der sehr viele Programme geschrieben wurden, sowie UCSD-Pascal. Das auf anderen Computern verbreitete System MS-DOS arbeitet nicht auf dem Apple //c. CP/M ist nur mit zusätzlicher Hardware möglich.

Die Eigenschaften der verschiedenen Betriebssysteme sind in Kapitel 2 beschrieben.

1.8 Die Unterschiede gegenüber dem Apple //e

Dieser Abschnitt ist vor allem wichtig für diejenigen, die schon mit dem Apple //e oder Vorgängermodellen gearbeitet haben, und diejenigen, die für den Apple //e geschriebene Programme verwenden wollen.
Der Apple //c ist auf 128 kB RAM Speicher begrenzt. Dies ist jedoch für die allermeisten Benutzer keine Einschränkung. In den USA wird eine Karte zum Einbau in den //c angeboten, die einen zusätzlichen Z80-Prozessor *und* 512 kB RAM besitzt (Z-RAM von Applied Engineering).
Ganz selten kann es vorkommen, daß manche bei Apple //e-Programmen dargestellte Zeichen (inverser Text) auf dem Apple //c anders dargestellt werden und zu Verwirrung führen. Die Sonderzeichen werden für Mausprogramme verwendet; sie sind im Anhang 7 dargestellt.

Da der Apple //c keine „Slots", also Steckplätze für Erweiterungskarten, besitzt, die der Apple //e hat, können nur Programme verwendet werden, die nur solche Anschlüsse verwenden, die auch im Apple //c eingebaut sind. Dies sind vor allem:

80 Zeichen-Karte mit 64 kB Speicher in Slot 3
Serielles Interface in Slot 1
Serielles Interface in Slot 2
Anschluß für Maus, Paddles oder Joystick in Slot 4
je ein Diskettenlaufwerk in Slot 6 und 7

Bemerkung: Diese Anschlüsse sind in die Platine des Apple //c integriert. Der Computer selbst behandelt und bezeichnet sie als „Slots", da er aus dem Apple //e entwickelt wurde, der solche Slots besitzt. Slot 5 ist „leer" bzw. nicht vorhanden.

Wenn Software verwendet werden soll, die für den Apple //e gedacht ist, muß man sich vorher überzeugen, ob diese Programme nicht Zusatzhardware, also Erweiterungskarten oder ähnliches benötigen, die auf dem Apple //c nicht vorhanden ist.
In einem solchen Fall laufen die Programme auf dem Apple //c entweder überhaupt nicht, oder nur die Programmteile, die die Zusatzhardware nicht benutzen.

Nicht geschützte, lesbare Programme für den Apple //e oder andere Apple // Computer können durch oft nur geringe Änderungen an den Apple //c angepaßt werden, soweit sie nicht sowieso schon auf ihm laufen.

Der Apple //c besitzt keinen parallelen Ausgang. Da aber viele Drucker nur parallele Anschlüsse besitzen, ist bereits Zusatzhardware erhältlich, die in einen der seriellen Anschlüsse des Apple //c gesteckt werden kann und selbst wieder einen parallelen Ausgang besitzt.
Für serielle Anschlüsse hat sich noch keine Norm durchgesetzt. Es kann also Geräte geben, die zwar über einen seriellen Anschluß verfügen, trotzdem aber nicht an den Apple //c angeschlossen werden können. Näheres dazu steht in Kapitel 5.
Am Apple //c können keine externen Tastatur angeschlossen werden, die für den Apple //e Computer angeboten werden, weil der interne Anschluß der Tastatur an die Computerplatine nicht zugänglich ist.
Der Apple //c besitzt andere ROM Routinen (fest eingespeicherte Maschinenprogramme) als der Apple //e. Das ist nur für Benutzer wichtig, die in Assembler programmieren wollen oder solche Routinen aufrufen (siehe Anhang).
Programme, die noch unter den älteren Betriebssystemen DOS 3.2 oder DOS 3.1 laufen, kann der Apple //c nicht verarbeiten. Diese Programme können aber mit Hilfprogrammen in DOS 3.3 Programme umgewandelt werden, die auf dem Apple //c laufen.

Kapitel 2

Die Bedienung des APPLE //c

2.1 Inbetriebnahme

Das mit dem Computer mitgelieferte Benutzer-Handbuch beschreibt sehr gut die Vorgehensweise bei der erstmaligen Inbetriebnahme der Maschine. Falls die beschriebene Vorgehensweise nicht funktionieren sollte, liest man am besten die Anleitung nochmals durch, schaltet alles aus und beginnt von neuem. Kurz zusammengefaßt sieht das so aus:

1. Prüfen, ob alle Kabel richtig angeschlossen sind.
   (Netzkabel von Computer und Monitor/Fernseher)
2. Monitor/Fernseher einschalten
3. Diskette (z.B. „Eine Einführung") in das eingebaute Diskettenlaufwerk einlegen.
4. Computer an der Rückseite einschalten.

Normalerweise sollte jetzt das Diskettenlaufwerk anfangen zu lesen und auf dem Monitor eine Meldung erscheinen. Läuft das eingebaute Laufwerk nicht an, so hat der Computer höchstwahrscheinlich keinen Strom. In diesem Fall sollten Sie alles wieder ausschalten, die Kabel prüfen und von vorne beginnen.

Arbeitet das Laufwerk einige Sekunden und bleibt dann stehen, ohne daß auf dem Monitor eine Meldung erscheint, dann ist der Monitor ohne Strom oder das Verbindungskabel vom Monitor zum APPLE //c ist nicht richtig angeschlossen.

Beim Betrieb mit Fernseher kann auch der PAL-Modulator nicht in Betrieb sein oder der Fernseher ist auf die falsche Frequenz eingestellt. Im letzteren Fall muß man die Computersignale am Fernseher suchen, so wie man beim Fernsehen einen anderen Sender sucht. Beim Betrieb eines Fernsehers sollte man den Computer mit dem Schalter 80/40 auf 40 Zeichendarstellung (pro Zeile) schalten.

Erscheint auf dem Monitor eine Meldung, so ist das ein gutes Zeichen, denn aus der Meldung kann man sehr viel ablesen. Erscheint beispielsweise die Meldung

APPLE //c

Check Disk Drive.

,so ist entweder keine Diskette eingelegt worden, oder das Diskettenlaufwerk ist nicht richtig verschlossen.
Erscheint die Meldung

Unable to load ProDOS

, so handelt es sich um eine ProDOS Datendiskette, nicht um eine ProDOS Start- oder Boot-Diskette. Dann sollte man die Prozedur mit einer anderen Diskette wiederholen und prüfen, ob die Diskette mit der Beschriftung nach oben eingelegt wurde.

Im Normalfall meldet sich der APPLE //c mit irgend einem Kommentar am Bildschirm. Dieser Kommentar läßt Schlüsse auf das Betriebssystem oder das Programm zu.
Grundsätzlich ist zu unterscheiden, ob sich der Computer aus dem Betriebssystem meldet oder nicht. Die für den APPLE //c wichtigsten Betriebssysteme ProDOS oder DOS 3.3 melden sich wie folgt:

```
PRODOS BASIC 1.0
COPYRIGHT APPLE, 1983
```

```
APPLE II

DOS VERSION 3.3  SYSTEM MASTER

JANUARY 1, 1983

COPYRIGHT APPLE COMPUTER,INC. 1980,1982
```

Erscheint etwas anderes auf dem Bildschirm, so handelt es sich vermutlich um ein Programm. Im Normalfall sollte ein Programm seinen Namen angeben und den Benutzer begrüßen bzw. ihm irgendwelche Informationen oder Anweisungen geben.
Bei einer sogenannten Autostart-Diskette wird ein vorhandenes Programm selbsttätig gestartet. Das ist im Allgemeinen bei käuflichen Programmen der Fall.

Erscheint eine der beiden Betriebssystem-Meldungen auf dem Bildschirm, so ist der Computer einsatzbereit, hat aber nichts zu tun. Daher wartet er auf eine Eingabe des Benutzers über die Tastatur.

Alle Anweisungen, Information oder sonstige Eingaben müssen mit dem Drükken der „Return-Taste" abgeschlossen werden, damit der Computer weiß, daß die Eingabe beendet ist und daß er die Eingabe irgendwie bearbeiten soll.

Eine Ausnahme davon bilden „ESC-4" und „ESC-8" (kurzes Drücken der Tasten ESCAPE und 4 bzw. 8). Damit kann in beiden Betriebssystemen sehr schnell zwischen 40 und 80 Zeichendarstellung pro Zeile hin und her geschaltet werden.

2.2 Inhaltverzeichnis der Disketten

Links auf dem Bildschirm befindet sich eine eckige Klammer] oder ein Ü, wenn Sie den deutschen Zeichensatz eingeschaltet haben. Diese Klammer wird in Zukunft immer wieder auftauchen und dem Benutzer deutlich machen, daß der APPLE //c auf eine Eingabe wartet.

Wir können dem Computer z.B. den Befehl „CATALOG", gefolgt von der „Return-Taste" (rechts), eingeben. Dieser Befehl veranlaßt den APPLE //c, auf der im Laufwerk liegenden Diskette das Inhaltsverzeichnis zu lesen und am Bildschirm darzustellen.
Wenn Sie eine Diskette eingelegt haben, die sich mit „DOS3.3" gemeldet hat, so müssen Sie alle Anweisungen an den APPLE //c groß schreiben, sonst kann sie der APPLE //c nicht verstehen und protestiert mit der Meldung „SYNTAX ERROR". Anweisungen unter ProDOS können auch kleingeschrieben werden, der Befehl „CATALOG" kann sogar mit „CAT" abgekürzt werden (das bringt dann ein reduziertes Inhaltsverzeichnis für 40 Zeichen Darstellung).

Ein Inhaltsverzeichnis unter DOS 3.3 sieht beispielsweise so aus:

```
ÜCATALOG

DISK VOLUME 254

 A 002 HELLO
*B 020 FID
*B 003 CHAIN
*I 009 COPY
*A 003 MASTER
*B 009 MASTER CREATE
*B 003 COPY.OBJO
*B 006 LOADER.OBJO
*A 003 FILEM
*A 003 START13
*B 007 BOOT13
*A 003 CONVERT13
 A 002 DEZDUAL
 A 002 TEXTSAVE
```

Ein Inhaltverzeichnis mit dem Befehl „CATALOG“ unter ProDOS sieht beispielsweise so aus:

```
Ü
ÜCATALOG

/BP2

 NAME            TYPE  BLOCKS  MODIFIED          CREATED           ENDFILE SUBTYPE

 PRODOS          SYS     30    18-SEP-84  0:00   <NO DATE>          14848
*BASIC.SYSTEM    SYS     21    15-NOV-83  0:00   <NO DATE>          10240
 STARTUP         BAS      1    <NO DATE>         <NO DATE>             81
 A7              AWP     15     6-JUN-85  0:00    6-JUN-85  0:00     7153

BLOCKS FREE:  206     BLOCKS USED:   74     TOTAL BLOCKS:  280
```

Das Inhaltverzeichnis zeigt eine Reihe von Namen mit zusätzlichen Informationen. Auf der Diskette werden die Informationen in verschiedene Abschnitte unterteilt, die alle einen Namen haben. Diese Abschnitte heißen „File“, d.h. Haufen. Diese Files können ganz unterschiedlichen Inhalt haben und auch unterschiedlich groß sein.
Das im Inhaltverzeichnis bei manchen Files ganz links angegebene Sternchen gibt an, daß das File vor Überschreiben geschützt ist.

*A 002 RANDOM (schreibgeschütztes File)

Anschließend steht im ProDOS Inhaltsverzeichnis der Filename (bei DOS 3.3 ganz rechts).
Die Größe eines Files ist beim ProDOS CATALOG zweimal angegeben; unter der Bezeichnung BLOCKS ist der vom File auf der Diskette belegte Platz in Blocks angegeben (1 Block = 512 Bytes) und unter der Bezeichnung ENDFILE direkt in Bytes.
Unter DOS 3.3 wird nur die Anzahl belegter Sektoren vor dem Filenamen dargestellt.
Die volle Breite des Inhaltsverzeichnisses bei ProDOS wird nur bei Eingabe von „CATALOG" wiedergegeben (dazu wird zweckmäßigerweise die 80-Zeichendarstellung benutzt). Wird das Inhaltsverzeichnis durch die Eingabe von „CAT" befohlen, so wird nur NAME, TYPE und BLOCKS angegeben. Dies ist vor allem für die Verwendung der 40 Zeichendarstellung gedacht.
BASIC-Programme im Inhaltsverzeichnis werden mit „BAS" bezeichnet (unter ProDOS), bzw. mit „A" ganz links (unter DOS 3.3).

2.3 Betriebssysteme

Die Art der Datenaufzeichnung auf die Disketten wird durch das Betriebssystem bestimmt. Das Betriebssystem ist ein Programm in Maschinensprache, welches zu einem kleinen Teil ständig im ROM (Read Only Memory) des Apple enthalten ist; z.B. die Information, daß der APPLE //c, sobald er eingeschaltet wird, im eingebauten Diskettenlaufwerk eine Diskette suchen und von dieser dann den (hoffentlich) vorhandenen Rest des Betriebssystems lesen soll.
Wenn dieser Rest (der größte Teil) des Betriebssystems eingelesen ist, dann ergibt z.B. der Befehl „CATALOG" einen Sinn, d.h. der Befehl wird als zulässig erkannt und ausgeführt. Das Einlesen des Betriebssystems von der Diskette wird in der Fachsprache „Booten" genannt.

Es ist durchaus möglich, mit dem APPLE //c zu arbeiten, ohne von einer Diskette gelesen zu haben (ohne Betriebssystem). Legt man beim Einschalten des Computers keine Diskette ein, so erscheint die Fehlermeldung „Check Disc Drive.". Dieser Meldung muß man jedoch nicht folgen, sondern kann durch Eingabe von Control-C (gleichzeitiges Drücken der Tasten CONTROL und C, gefolgt von Return) die Fehlermeldung ignorieren.
Der Computer ist dann bereit, Befehle auszuführen, und tut dies durch die Ausgabe einer eckigen Klammer kund. Anschließend kann man Programme in

BASIC eingeben und ausführen lassen (die Sprache BASIC ist immer im APPLE vorhanden und muß deshalb nicht von der Diskette gelesen werden). Da der Computer aber kein Betriebssystem von der Diskette gelesen hat, ist für ihn der Betriebssystembefehl „SAVE Filename" unverständlich. SAVE Filename bedeutet eigentlich, daß das vorher eingegebene Programm mit dem Namen „Filename" auf die Diskette geschrieben werden soll. Ohne Betriebssystem ist der Computer also nicht allzuviel wert, da alles wieder verloren geht, was eingegeben wurde, sobald er ausgeschaltet wird.

Die Speicherung des Betriebssystems auf der Diskette hat den Nachteil, daß man nach dem Einschalten einige Sekunden warten muß, bis der Computer arbeitsbereit ist. Da sich die Software (und ein Betriebssystem ist ein Programm und somit Software) relativ schnell entwickelt, ist es aber möglich, auf dem Computer immer das neueste oder am besten geeignete Betriebssystem zu verwenden, was für den APPLE //c auf Diskette erhältlich ist.

Am Wichtigsten sind die beiden Betriebssysteme der Firma APPLE selbst, das neuere ProDOS und sein Vorgänger DOS 3.3. Der Name DOS ist die Abkürzung von Disc-Operating-System (Disketten-Betriebssystem). Weiterhin sind noch andere Betriebssysteme für den Apple //c erhältlich z.B. UCSD Pascal, Diversi DOS etc.

Beide Betriebssysteme (DOS und ProDOS) sind sich sehr ähnlich, die meisten Befehle sind identisch. Der eigentliche Unterschied für den Benutzer betrifft die Befehle zum Datenaustausch zwischen dem RAM Speicher des APPLE und den Disketten.
Im Folgenden wird das Betriebssystem ProDOS erläutert. Anschließend werden die Unterschiede zum DOS 3.3 erklärt:

2.3.1 ProDOS

ProDOS (Professional Disk Operating System) ist eine Weiterentwicklung des älteren DOS 3.3 und stammt genaugenommen vom SOS (Sophisticated Operating System) des mittlerweile nicht mehr produzierten Apple /// ab. ProDOS enthält gegenüber DOS 3.3 einige wesentliche Verbesserungen, zum Teil neue Befehle und schnellere Diskettenroutinen. Alle Vorteile von ProDOS hier zu beschreiben ist nicht möglich, dafür sind spezielle Bücher und Manuals erhältlich (siehe Kapitel 6). An dieser Stelle sollen einige für den BASIC-Programmierer nützliche Hinweise gegeben werden.

2.3.1.1 Datum und Uhrzeit unter ProDOS

Im ProDOS Inhaltsverzeichnis (CATALOG) ist oft Datum und Uhrzeit angegeben. Im Apple //c ist keine Uhr vorhanden. Deshalb können diese Informationen vom Benutzer eingegeben werden. Das folgene Programm fragt Datum und Uhrzeit ab und legt diese verschlüsselt im RAM Speicher ab:

```
 10  REM  DATUMPROGRAMM setzt Datum und Uhrzeit für CATALOGs

 20  HOME
 30  INPUT "Bitte Jahr eingeben : 19";J: POKE 49041,J * 2
 40  INPUT "Bitte Monat eingeben   :";M: IF M > 7 THEN
     POKE 49041, PEEK (49041) + 1
 50  INPUT "Bitte Tag eingeben     :";T
 60  IF M > 7 THEN  POKE 49040,(M - 8) * 32 + T: GOTO 80
 70  POKE 49040,M * 32 + T
 80  INPUT "Bitte Uhrzeit (Stunden) ;"S: IF S > 12 THEN
     POKE 49043,S - 12: GOTO 100
90  POKE 49043,S
100  INPUT "Bitte Uhrzeit (Minuten) ";M: POKE 49042,M
110  INPUT "Alles richtig eingegeben ? J/N";X$:
      IF X$ = "N" OR X$ = "n" THEN  HOME : GOTO 30
```

Das Programm fängt keine Fehler ab (es prüft z.B. nicht, ob die Monatszahl kleiner als 13 ist). Es kann unter dem Namen STARTUP auf einer ProDOS Diskette des Benutzers gespeichert werden (SAVE STARTUP). STARTUP wird dann bei jedem Booten (Neustart) ausgeführt (wenn vorhanden) und damit das Datum gesetzt. Auf der ProDOS USERS DISK ist eine ähnliche ROUTINE vorhanden.

Anmerkung: Unter ProDOS sollte in jedem Fall ein STARTUP-Programm zu Beginn (beim Booten) ausgeführt werden, egal welchen Inhalts.

2.3.1.2 Filestruktur

ProDOS erlaubt hierarchische Filestrukturen. Das bedeutet, mehrere Files können unter einer Subdirectory zusammengefaßt werden, einer Art zusätzlichem Gesamtfilenamen. Wird kein Subdirectory verwendet, so werden Filename und (falls verwendet) Diskettenname (auch Volumename) durch den Schrägstrich getrennt dargestellt.

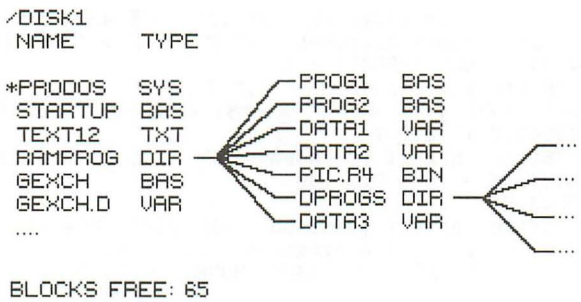
Beispiel: PRODOS2/TEXTFILE

PRODOS2 ist der Diskettenname, TEXTFILE ist der Filename.

Existiert ein Subdirectory, so wird es zwischen Diskettenname und Filename angegeben:

Beispiel: PRODOS2/DIR.TEST/TEXTFILE

Subdirectories können in mehreren Ebenen (geschachtelt) definiert werden; aus Gründen des Übersichtlichkeit ist das nicht empfehlenswert.



Subdirectories unter ProDOS

Subdirectories erkennt man im Inhaltverzeichnis an der Bezeichnung DIR; z.B.:

```
ÜCATALOG

/BP2

 NAME            TYPE  BLOCKS  MODIFIED          CREATED           ENDFILE SUBTYPE

 PRODOS           SYS      30  18-SEP-84  0:00  <NO DATE>           14848
*BASIC.SYSTEM     SYS      21  15-NOV-83  0:00  <NO DATE>           10240
 STARTUP          BAS       1  <NO DATE>        <NO DATE>              81
 A7               AWP      15   6-JUN-85  0:00   6-JUN-85  0:00      7153
 RAMFILES         DIR       1  <NO DATE>        <NO DATE>             512

BLOCKS FREE:  189     BLOCKS USED:   91     TOTAL BLOCKS:  280
```

Um den Inhalt des Subdirectories darzustellen, muß es nach der Eingabe „CATALOG“ (oder CAT) angegeben werden:

```
ÜCATALOG/BP2/RAMFILES

RAMFILES

 NAME            TYPE  BLOCKS  MODIFIED        CREATED         ENDFILE SUBTYPE

 RAM.READ         BAS       7  <NO DATE>       <NO DATE>          2968
 RAM.DAT          VAR       4  <NO DATE>       <NO DATE>          1193
 RAM.DAT2         VAR       5  <NO DATE>       <NO DATE>          1803

BLOCKS FREE:  189     BLOCKS USED:   91     TOTAL BLOCKS:  280
```

Auf das ProDOS Directory kann wie auf ein anderes File zugegriffen werden, wie folgendes Beispiel zeigt:

```
1    REM Stellt die BASIC-Programme des Inhaltsverzeichnisses
10   REM einer ProDOS - Diskette dar.
20   HOME
30   INPUT "Bitte ProDOS Diskettennamen eingeben: ";F$
40   IF  LEFT$ (F$,1) <  > "/" THEN F$ = "/" + F$
50   HOME : PRINT "Die Basic Programme auf dieser
     Diskette heissen:"
60   PRINT
70   D$ =  CHR$ (4)
80   ONERR  GOTO 140
90   PRINT D$;"OPEN"F$",TDIR"
100   PRINT D$;"READ"F$
110   FOR I = 1 TO 50
120   INPUT A$: IF  MID$ (A$,18,3) = "BAS" THEN
      PRINT  MID$ (A$,2,16)
130   NEXT I
140   PRINT D$"CLOSE"
150   POKE 216,0
160   END
200   REM
210   REM   BEISPIEL FUER A$:
220   REM  *STARTUP          BAS        7
      1-OCT-83  0:00   1-OCT-83  0:00     2718
230   REM  Name : MID$(A$(I),2,16)
240   REM  TYPE : MID$(A$(I),18,3)
250   REM  BLOCKS : MID$(A$(I),26,3) usw.
```

Weil die Angabe des gesamten Diskettennamens plus Directoryname (ev. plus weiterer Sub-Subdirectory Name) plus Filename bei mehrmaligem Gebrauch aufwendig ist, kann unter ProDOS der Prefix definert werden.

PREFIX/DISK23/DIR.TEST

setzt den Prefix auf den Diskettennamen DISK23 und den Subdirectorynamen DIR.TEST. Ist der Prefix definiert, so wird er bei jedem Filezugriff oder bei

CATALOG verwendet. Der Prefix wird bei Bedarf entweder überschrieben oder gelöscht:

PREFIX.S6.D1	setzt den Prefix auf das eingebaute Diskettenlaufwerk.
PREFIX/	löscht den Prefix.
PREFIX	zeigt den aktuellen Prefix an.

ProDOS ermöglicht das Chaining (Verketten) von Programmen. Das bedeutet, ein Programm kann ein weiteres Programm aufrufen, welches die Daten des alten Programms übernimmt. Mehr dazu im Kapitel 3.

2.3.2 DOS 3.3

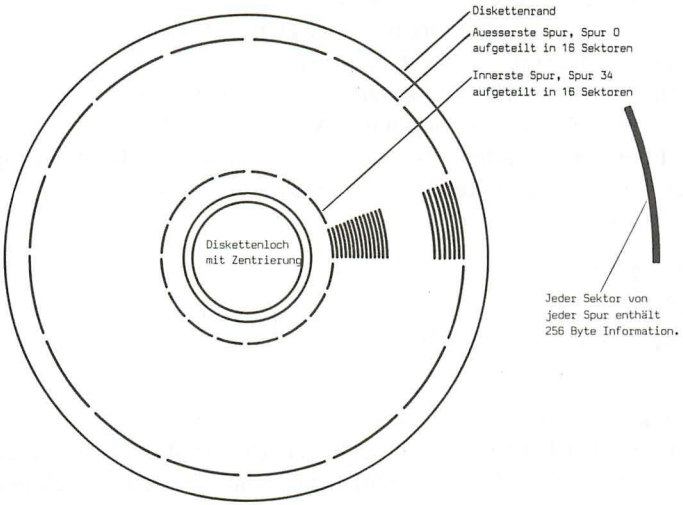
DOS 3.3 ist das „alte" Betriebssystem. Es ist zwar (im Filezugriff) langsamer und weniger leistungsfähig als ProDOS, benötigt andererseits weniger Speicherplatz. Viele bestehende Programme sind unter DOS 3.3 geschrieben worden. Mit den ProDOS System Dienstprogrammen lassen sich DOS 3.3 Programme (und Daten) in ProDOS Programme (und Daten) umwandeln. Es gibt allerdings (wenige) Programme, die unter ProDOS dann nicht laufen.
Für die in diesem Buch angegebenen Programme spielt es keine Rolle, unter welchem Betriebssystem sie laufen; Ausnahmen, die z.B. spezielle ProDOS Befehle demonstrieren, sind besonders gekennzeichnet.
Chaining, Datum und hierarchische Filestruktur sind unter DOS 3.3 nicht möglich.
Die Befehle des Betriebssystems DOS 3.3 sind in Kapitel 3 beschrieben.
Unter DOS 3.3 sind ebenfalls Autostart-Disketten möglich. DOS 3.3 führt beim Booten ein Programm namens „HELLO" aus, falls vorhanden.
DOS 3.3 unterscheidet nicht Boot-Disk und Daten-Disk; siehe hierzu Abschnitt 2.4.1.

2.3.3 UCSD Pascal

UCSD Pascal sei hier nur der Vollständigkeit halber erwähnt. Dieses Betriebssystem erlaubt die Programmierung in Pascal (und Fortran). Für Basic oder Assemblerprogrammierer hat es keine Bedeutung. Die Sprache Pascal wird in Kapitel 4 angeschnitten.

2.4 Disketten formatieren

Die Information auf den Disketten wird in 35 Spuren (konzentrische unsichtbare Ringe) gespeichert, die in jeweils 16 Sektoren aufgeteilt sind. Jeder dieser 560 Sektoren enthält 256 Bytes Informationen; auf einer Diskette haben somit 560 * 256 = 143.360 Bytes Platz.
1024 Bytes werden auch als Kilobyte (kB) bezeichnet, ähnlich wie beim Kilogramm.



Neue Disketten sind jedoch leer, d.h. sie enthalten keinerlei Informationen, auch keine darüber, wo sich Sektoren befinden und wo ungenutzter Zwischenraum ist. Um dem Computer zu ermöglichen, auf die Diskette zu schreiben, müssen vorher die Sektoren aufgeteilt werden und ein Inhaltsverzeichnis auf der Diskette angelegt werden, aus welchem der Computer ablesen kann, an welchen Stellen der Diskette noch Platz ist. Diesen Vorgang nennt man formatieren.
Die dafür notwendige Prozedur hängt vom verwendeten Betriebssystem ab.

2.4.1 Formatieren unter DOS 3.3

Zum Formatieren unter DOS 3.3 dient der Befehl „INIT".

Zunächst wird mit einer DOS 3.3 Diskette oder einer Kopie davon „gebootet"; d.h. der Computer wird bei eingelegter Diskette eingeschaltet oder durch gleichzeitiges Drücken der Tasten „OPEN-APPLE", „CONTROL" und „RESET" rebootet.

Bei der Formatierung werden auf der leeren Diskette die vorgesehenen 35 Spuren erzeugt, sowie ein Inhaltverzeichnis, das Betriebssystem DOS 3.3 und ein erstes Programm-File abgelegt. Dieses File bekommt den direkt nach dem Befehl „INIT" eingegebenen Filenamen.
Beim „Booten" sucht das Betriebssystem DOS 3.3 nach einem File namens „HELLO" und führt dieses aus, falls vorhanden.
Durch ein geeignetes HELLO-Programm kann man so Disketten erzeugen, welche sofort nach dem Start bestimmte Aufgaben ausführen, wie zum Beispiel weitere Programme starten. Wir wollen hier ein kurzes HELLO-Programm erzeugen, welches die Diskette selbst vorstellt.
Dazu geben wir ein:

```
10 REM  PROGRAMM HELLO
20 HOME
30 PRINT "DOS 3.3        Formatiert am 11.2.85"
40 PRINT CHR$(4);"CATALOG"
50 NEW
```

Natürlich kann auch ein anderes Datum eingegeben werden.
Mit dem Befehl „LIST" wird das Programm noch einmal zur Kontrolle auf dem Bildschirm dargestellt. Falsch geschriebene Zeilen werden einfach nochmals neu eingegeben.
Anmerkung: Das beim Formatieren auf die Diskette geschriebene File muß nicht zwangsläufig einen Inhalt haben, es kann auch leer sein.
Anschließend legen wir die zu formatierende Diskette in das eingebaute Laufwerk ein und geben den Befehl:

INIT HELLO

Das Laufwerk macht beim Formatieren etwa 1 Minute lang stärkere Geräusche als bei normalem Betrieb. Durch wiederholtes Wechseln der Disketten und Eingabe von „INIT HELLO" lassen sich mehrere Disketten hintereinander formatieren, doch **VORSICHT**:

Auch schon beschriebene Disketten werden ohne Rücksicht auf Verluste formatiert und die darauf vorhandenen Daten zerstört. Schreibgeschützte Diskette (mit verschlossener Schreibkerbe) können nicht formatiert werden.

Die erfolgreiche Formatierung kann man leicht durch „Booten" mit der neuen Diskette überprüfen.
Beim Formatieren unter DOS3.3 wird das momentan im Computer vorhandene Betriebssystem auf die Diskette geschrieben. Dabei geht Speicherplatz auf der Diskette verloren. Das auf der Diskette gespeicherte Betriebssystem wird beim CATALOG nicht angegeben, belegt aber 3 Spuren (ca. 12 kB).

CATALOG

DISK VOLUME 254

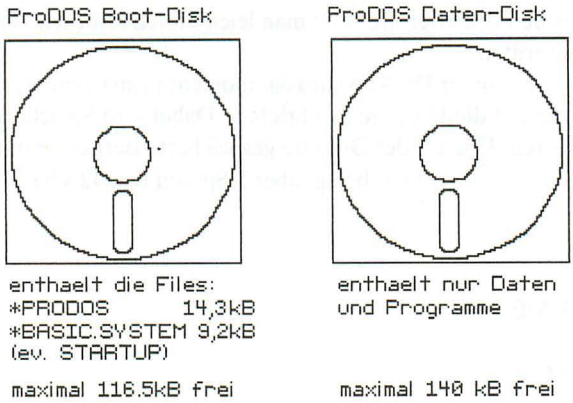
A 002 HELLO

Durch die Eingabe der Volumenummer beim „INIT"-Befehl kann der Diskette eine Zahl zwischen 1 und 254 zugeordnet werden (z. B. INIT HELLO, V 117)

2.4.2 Formatierung unter ProDOS

Unter dem Betriebssystem ProDOS werden Disketten mit Hilfe der mitgelieferten System Dienstprogramme formatiert. Dazu ist es notwendig, neu zu booten. Es ist deshalb empfehlenswert, mehrere Disketten auf einmal zu formatieren.
Bei der Formatierung unter ProDOS wird der Diskette ein Volumename zugeordnet. Dieser Volumname dient dazu, die Diskette später im Betrieb durch den Computer zu identifizieren. Der Volumename wird deshalb am besten auf die Diskettenhülle geschrieben, um spätere Schwierigkeiten bei Aufruf von Programmen zu vermeiden.
Bei ProDOS werden Datendisketten und Boot-Disketten unterschieden. Datendisketten enthalten nur Programme und andere Daten, aber nicht das ProDOS Betriebssystem.
ProDOS Boot-Disketten enthalten immer die Files PRODOS und BASIC.SYSTEM (meist auch ein STARTUP File). Ein Beispiel für ein STARTUP Programm, welches Datum und Uhrzeit setzt, ist in Abschnitt 2.3.1.1 gelistet.
Eine ProDOS Boot-Diskette wird aus einer normalen formatierten ProDOS (Daten-) Diskette erzeugt, indem mit dem Programm FILER (welches auch das

Formatieren vornimmt) die Files PRODOS und BASIC.SYSTEM von einer anderen ProDOS Boot-Diskette auf die neue Boot-Diskette kopiert werden.

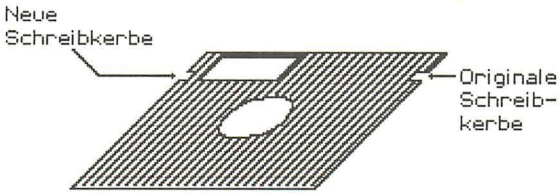


2.5 Die Handhabung der Disketten

Die auf den Disketten vorhandenen Informationen sind magnetisch gespeichert, ähnlich wie bei einem Tonband. Im Normalfall bleibt diese Information sehr lange erhalten (mehrere Jahre). Durch unsachgemäße Behandlung können aber Daten verloren gehen, d.h. für das Diskettenlaufwerk unlesbar werden.
Insbesondere muß man darauf achten, die Disketten vor Staub, Hitze, Magneten und Knicken zu schützen.
Es ist empfehlenswert, die Disketten nicht in die Sonne oder auf die Heizung zu legen. Besonders vorsichtig sollte man mit Scheren sein, da diese oft magnetisch sind und die Daten auf den Disketten beschädigen können. Wenn die Disketten nicht gebraucht werden, sollten sie immer in der Papierhülle bleiben und in Diskettenboxen aufbewahrt werden. Fingerabdrücke auf der Diskette selbst sind möglichst zu vermeiden.
Ab und zu kann es vorkommen, daß bestimmte Disketten beim Lesen oder Schreiben stärkere Raschelgeräusche entwickeln als normal. Diese Disketten sollte man nicht verwenden (die darauf vorhandenen Informationen auf eine andere Diskette kopieren).
Von allen wichtigen Daten auf Disketten werden am besten Kopien angefertigt (je wichtiger die Daten sind, umso öfter). Besonders beim Eingeben großer

Datenmengen (Adresslisten, lange Briefe, Programme) ist es zu empfehlen, etwa alle halbe Stunde die Informationen auf Diskette zu speichern.

Praktisch alle Disketten können auf beiden Seiten verwendet werden. Da die von APPLE angebotenen Laufwerke nur einen Schreib-Lese-Kopf (er liest auf der Unterseite der Disketten) besitzen, muß die Diskette zur Benutzung der Rückseite umgekehrt in das Laufwerk gelegt werden. Dazu ist eine Vorbereitung nötig: Die Disketten haben am rechten Rand eine Kerbe, die einen Schalter im Laufwerk betätigt, der das Schreiben auf die Diskette ermöglicht. Zum Benutzen der Rückseite der Diskette muß man am linken Rand auf gleicher Höhe ebenfalls eine solche Kerbe anbringen, die bei umgekehrter Diskette wieder rechts ist.



Anordnung der Schreibkerbe fuer die Verwendung der Rueckseite

Die Kerbe kann mit einem Klebstreifen (wird mit den Disketten mitgeliefert) wieder überklebt werden. Dann ist sie schreibgeschützt, darauf vorhandene Informationen können nicht mehr versehentlich gelöscht werden.
Disketten, die auf beiden Seiten verwendet werden, verschleißen schneller als normal, da der im Laufwerk auf der Diskettenoberseite vorhandene Andruckfilz über die vorhandenen Informationen schleift und dort Abrieb verursacht. Diese Methode ist deshalb nur bei großen Datenmengen zu empfehlen, bei denen die Kosten für die Disketten ins Gewicht fallen.

2.6 Die Pflege des Computers

Der Computer ist im Prinzip wartungsfrei. Wichtig ist nur, daß er im Betrieb nicht zu warm wird, also nicht in der prallen Sonne steht und die Lüftungsschlitze nicht abgedeckt sind.

Der Computer darf außerdem nicht naß werden, vor allem darf er nicht eingeschaltet werden, falls er einmal feucht sein sollte.
Die Reinigung der Maschine beschränkt sich auf Abstauben und gelegentliches Abwischen mit einem leicht feuchten Tuch. Die Tastatur und die Anschlüsse auf der Rückseite werden besser nicht feucht abgewischt.

Kapitel 3

APPLESOFT BASIC

In diesem Kapitel wird die Sprache APPLESOFT BASIC beschrieben; die Befehle werden nicht alphabetisch erklärt, sondern in einer Reihenfolge, die es dem Benutzer erlaubt, die Sprache durch einfache anschauliche Anwendungen zu erlernen. Die Erklärung der Befehle baut in der Regel auf den vorher behandelten Befehlen auf.
Wenn Sie dieses Kapitel durchlesen und gleichzeitig die angegebenen Programme auf dem Computer laufen lassen wollen, so sollten sie eine leere formatierte Diskette zur Verfügung haben.

BASIC ist eine Programmiersprache. Sie ermöglicht es dem Benutzer, dem Computer Befehle in Form von Einzelanweisungen oder Programmen zu geben, die in der Darstellung für den Menschen geeignet sind. (Die eigentliche Maschinensprache ist für den Menschen praktisch nicht direkt lesbar.)
Wie jede Programmiersprache besteht auch BASIC aus Befehlen oder Anweisungen, bei deren Eingabe der Computer bestimmte festgelegte „Arbeiten" ausführt. Die Aneinanderreihung von Anweisungen nennt man Programm.
Die Sprache BASIC (Abkürzung für Beginners All Purpose Instruction Code, Universeller Instruktions Code für Anfänger) existiert in vielen verschiedenen Ausführungen, Dialekte genannt. Der Apple //c spricht den Dialekt „APPLESOFT", eine Version, die speziell für die Computer der Apple // – Familie geschrieben wurde.
Bei manchen Programmierern hat BASIC einen nicht allzu guten Ruf; dies vor allem deshalb, weil BASIC den Anwender nicht zur strukturierten Programmierung anhält, was bei großen Programmen und bei der Fehlersuche nachteilig sein kann.
Ältere BASIC-Versionen, vor allem Standard-BASIC, hatten auch nicht den Wortumfang und die Leistungsfähigkeit der jetzt gebräuchlichen Dialekte; dies kann auch zum schlechten Image beigetragen haben.
Ein eindeutiger Nachteil von BASIC sind die verschiedenen Dialekte. Ein in APPLESOFT geschriebenes Programm wird nur auf Computern der Apple // –

Familie, Kopien davon und auf dem Apple /// laufen. Computer anderer Fabrikate akzeptieren APPLESOFT nicht.

Die BASIC-Version APPLESOFT der Firma Microsoft wird von APPLE mit dem Computer mitgeliefert. Die Vorgängerversion INTEGER-BASIC läuft auch auf dem Apple //c, aber nur unter DOS 3.3 oder älteren Betriebsystemen; ProDOS unterstützt diese BASIC Version nicht mehr. INTEGER-BASIC hat in den letzten Jahren stark an Bedeutung verloren und wird sehr wenig angewendet; um den Leser nicht durch zusätzliche Informationen über Unterschiede vom INTEGER-BASIC zum APPLESOFT-BASIC zu verwirren, werden in diesem Buch diese Differenzen nicht behandelt.

Die Bereitschaft des Computers zur Arbeit in BASIC erkennt man am „Cursor“:

] oder Ü.

Beim Erscheinen dieses Zeichens ist der Apple //c in der Betriebsart „Einzelanweisung“. Er wartet dann auf die Eingabe eines Befehls (in BASIC) und führt diesen nach dem Drücken der Return-Taste aus.
Dabei wird der eingegebene Befehl geprüft. Der Computer versucht dann, den „Sinn“ des Befehls zu verstehen. Diesen Vorgang nennt man „interpretieren“. Die Sprache BASIC besteht aus einem Programm, Interpreter genannt, welches den eingegebenen Befehl analysiert und wenn möglich ausführt. Versteht der Interpreter den Befehl nicht, so meldet er das dem Benutzer mit dem Hinweis „? SYNTAX ERROR“ und einem Piepston.

3.1 Die Einzelanweisungen

Befehle: **PRINT, ?**

In der Betriebsart „Einzelanweisungen“ führt der Computer nach dem Drücken von „RETURN“ den eingegebenen Befehl sofort aus. Wird die Anweisung

```
PRINT "Dies ist die erste Antwort"
```

eingeben, so erscheint auf dem Bildschirm das Ergebnis des Befehls:

Dies ist die erste Antwort

Nach Eingabe von Zeichenfolgen (Befehlen), die für den Computer unverständlich sind, antwortet dieser mit „? SYNTAX ERROR". Dann haben Sie sich vermutlich vertippt. Der Befehl darf höchstens 239 Zeichen lang sein, Leerzeichen werden mitgezählt. Dies sind etwa 3 Zeilen in 80-Zeichen Darstellung, bzw. etwa 6 Zeilen bei 40-Zeichen Darstellung. Kurz vor dieser Grenze von 239 Zeichen gibt der Apple Piepstöne von sich; wenn man mehr als 239 Zeichen eingibt, wird ein Backslash (umgekehrter Schrägstrich) ausgegeben und der Befehl ignoriert. Wenn man über das Zeilenende hinausschreibt, so wird in der nächsten Zeile vorne weitergemacht. Das hat auf die Ausführung des Befehls aber keinen Einfluß.
Der Apple läßt sich wie ein Taschenrechner verwenden. Als einzigen Unterschied muß man zusätzlich den Befehl „PRINT" eingeben, sonst wird zwar die Rechnung durchgeführt, aber das Ergebnis nicht ausgegeben. Da der Befehl „PRINT" sehr häufig benötigt wird, kann man ihn durch das Fragezeichen ersetzen. Die Befehle

PRINT 17 + 4 und ? 17 + 4

führen beide zum Ergebnis 21.

Bei der Eingabe von Befehlen müssen einzelne Teile nicht voneinander getrennt werden. Der Computer teilt sich den Text selbst in die für ihn verständlichen Worte auf. Die beiden folgenden Eingaben haben also gleiche Bedeutung:

```
PRINT12*12+6
150

PRINT 12 * 12 + 6
150
```

Die selbständige Trennung hat allerdings auch Nachteile. Manche Worte werden nicht so getrennt, wie der Benutzer das gerne hätte. Darauf komme ich später noch zurück.
Es können auch mehrere Befehle nacheinander eingegeben werden. Dazu werden die einzelnen Befehle durch Doppelpunkte getrennt:

```
? 200 * 5 : ? "Zweite Zeile"
1000
Zweite Zeile
```

Eine Reihe von Befehlen kann nicht in der Betriebsart „Einzelanweisungen" ausgeführt werden, sondern nur innerhalb eines Programms. Diese Befehle sind: APPEND, OPEN, POSITION, READ und WRITE; sie werden später erklärt.

3.2 Die Programmzeilen

Befehle: **RUN, LIST**

Die Betriebsart „Einzelanweisungen" ist dann etwas umständlich, wenn man eine Reihe von Befehlen ausführen lassen will. Dazu dient ein Programm.
Analog zu den Einzelanweisungen müssen die Befehle vor der ersten Ausführung eingetippt werden; im Unterschied zu dort wird zuerst eine Zeilennummer eingegeben und anschließend die Befehle. Die Zeilennummer dient dazu, einzelne Zeilen, aus denen das Programm gebildet wird, zu unterscheiden. Als Zeilennummern sind ganze Zahlen zwischen 0 und 63999 (unter ProDOS von 0 bis 65535) einschließlich möglich.
Wenn Sie beispielsweise die folgende Programmzeile eingeben

```
10 FOR I=1 TO 6: ? SPC(I); I:NEXT I
```

,so werden die drei Befehle nicht ausgeführt. Der Computer behält sie nur im Speicher als Programm. Das Programm wird gestartet durch den Befehl „RUN". Wenn Sie nun „RUN" eingeben, so erscheint:

```
1
 2
  3
   4
    5
     6
```

Das Programm bleibt auch nach der Ausführung durch den Befehl „RUN" erhalten. Mit dem Befehl „LIST" kann man das Programm wieder darstellen.

```
10  FOR  I = 1 TO  6 : PRINT SPC( I ) ; I : NEXT I
```

Die neue Darstellung heißt „Listing". Der Computer stellt das Programm etwas anders dar als eingegeben, weil er nach bestimmten Regeln Leerzeichen zwi-

schen den Befehlen ausgibt und das abgekürzte PRINT ausgeschrieben darstellt.
Die Programmzeile bleibt auch erhalten, wenn weitere Zeilen eingegeben werden. Nach der Eingabe von

```
5 ?"HALLO"
20 ?:?"ENDE"
```

und dem Befehl „LIST" erscheinen die Programmzeilen in der richtigen Reihenfolge. Sie werden vom Apple automatisch geordnet:

```
LIST

5   PRINT  "HALLO"
10   FOR I = 1 TO 6 : PRINT SPC( I ) ; I : NEXT I
20  PRINT: PRINT  "ENDE"
```

Die Einordnung in der richtigen Reihenfolge hat seinen Grund: Die Programmzeilen werden in aufsteigender Reihen ausgeführt, nicht in der Reihenfolge der Eingabe. Das Programm fängt also mit der Zeile mit der niedrigsten Zeilennummer an und hört mit der letzten auf. Diese Reihenfolge der Abarbeitung kann nur mit bestimmten Befehlen unterbrochen werden.
Der Befehl LIST kann durch Zeilennummern ergänzt werden:

LIST 5,10 : Listing von Zeile 5 bis Zeile 10 (einschließlich)
LIST ,10 : Listing von erster Zeile bis Zeile 10 (einschließlich)
LIST 10, : Listing von Zeile 10 bis zum Ende

Eigentlich sollten Sie das Programm schon soweit lesen können, daß Sie wissen, was es tut, wenn es gestartet wird.
RUN

```
HALLO
 1
  2
   3
    4
     5
      6
ENDE
```

Gibt man nun ein:

```
10 PRINT "neue Zeile":?
```

,so wird die alte Zeile 10 gelöscht und durch die neue ersetzt, was Sie mit „LIST" nachprüfen können. Beim Starten des Programms mit „RUN" erscheint dann:

HALLO
neue Zeile

ENDE

Statt „RUN" kann man auch „RUN 10" eingeben. Dann beginnt der APPLESOFT-Interpreter ab Zeile 10 zu lesen und das Programm macht folgendes:

RUN 10

neue Zeile

ENDE

Die Zeile fünf ist nicht ausgeführt worden. Trotzdem ist sie noch im Programm vorhanden.

Maximal 239 Zeichen können pro Programmzeile (nicht zu verwechseln mit Darstellungszeile) eingegeben werden, sonst werden die letzten Zeichen abgeschnitten.

Mit diesem Wissen ist es nun möglich, große Programme zu schreiben und ablaufen zu lassen.
Wird der Computer ausgeschaltet, so geht das Programm verloren. Um das zu verhindern, wird es auf Diskette gespeichert.

3.3 Programme auf Disketten speichern,laden und ändern

Befehle: **CAT, CATALOG, SAVE, LOAD, NEW, END, DELETE, LOCK, UNLOCK, VERIFY, RENAME, HOME.**

Die Disketten dienen zum Speichern von Daten aller Art. In diesem Kapitel wird beschrieben, wie Programme auf Disketten geschrieben werden können.

Die Speicherung von Daten anderer Art (Text, Binary-Daten) wird in einem späteren Abschnitt behandelt.

Ein im Computer vorhandenes APPLESOFT-Programm wird mit dem Befehl „SAVE Filename“ auf eine im Laufwerk eingelegte Diskette geschrieben. Die Bezeichnung „Filename“ ist die vom Benutzer gewählte Bezeichnung des Programms auf der Diskette. Nur unter dieser Bezeichnung kann das Programm wieder gelesen werden. Natürlich muß die Diskette richtig formatiert sein. (Wie man unter DOS 3.3 oder ProDOS formatiert, steht in Kapitel 2).

Das im obigen Abschnitt geschriebene Programm (oder ein anderes APPLESOFT-Programm) können Sie speichern (oder sichern), indem Sie die richtige Diskette ins Laufwerk legen und eingeben:

SAVE PROGRAMM1

Das Laufwerk beginnt zu arbeiten und das rote Licht rechts oberhalb der Tastatur leuchtet. Ist der Vorgang beendet und der Cursor erscheint wieder auf dem Bildschirm, so läßt sich mit „CAT“ (nur unter ProDOS) oder „CATALOG“ kontrollieren, ob das Programm gespeichert worden ist.

CAT

```
/PRODOS2

NAME          TYPE  BLOCKS  MODIFIED

PROGRAMM1     BAS        1  NO DATE

BLOCKS FREE: 272    BLOCKS USED: 8
```

Der Filename „PROGRAMM1“ muß dann in der Fileliste vorkommen (nicht unbedingt am Ende!).
Falls auf der Diskette ein ungeschütztes File (siehe „LOCK“) mit dem Filenamen „PROGRAMM1“ vorhanden war, so wird es durch den Befehl „SAVE PROGRAMM1“ überschrieben. Das ist vor allem bei der Programmentwicklung zu beachten!

Einen ähnlichen Sinn wie CATALOG hat der Befehl „VERIFY“. Dieser prüft unter DOS 3.3, ob das im Computer vorhandene Programm richtig auf die Diskette geschrieben wurde. Dazu gibt man ein:

VERIFY PROGRAMM1

Erscheint nach der Prüfung nur der Cursor auf dem Bildschirm, so ist alles in Ordnung. VERIFY vergleicht übrigens nicht das im Computer vorhandene Programm mit dem auf der Diskette, sondern liest das Programm auf der Diskette und vergleicht es mit einer zusätzlich gespeicherten Prüfsumme. Im Allgemeinen kann aber davon ausgehen, daß das Programm richtig abgespeichert wurde, falls nicht schon bei der Verwendung von SAVE eine Fehlermeldung auftrat. Der Befehl VERIFY muß deshalb nicht nach jedem SAVE benutzt werden.
VERIFY unter ProDOS prüft nur die Existenz eines Files, nicht aber seine Korrektheit. Dazu gibt es im FILER eine „BAD BLOCKS"-Routine.

Analog zum Befehl SAVE wird der Lesebefehl „LOAD" verwendet (LOAD Filename). Mit dem Befehl LOAD sollte man immer vorsichtig umgehen, denn jedes eventuell mühsam erarbeitete Programm wird durch „LOAD Filename" im Computer-Hauptspeicher gelöscht und durch das neugelesene ersetzt. Sie sollten also vor jedem LOAD prüfen, ob das im Computer eventuell vorhandene Programm zerstört werden darf. Wenn nicht, müssen Sie es vorher mit SAVE auf Diskette speichern.

Will man ein auf der Diskette vorhandene File PROGRAMM1 vor dem Überschreiben schützen, so geschieht das mit dem Befehl „LOCK Filename". LOCK bedeutet „Abschließen" und versieht das File „Filename" mit einer Markierung, welche es vor Überschreiben und Löschen schützt. Solche Files sind bei CATALOG vor dem Filenamen mit einem Stern (*) gekennzeichnet und können nur gelesen werden. Wichtige Files sollte man deshalb mit LOCK schützen.
Nach den Befehlen

LOCK PROGRAMM1
CAT

erscheint auf dem Bildschirm:

```
/PRODOS2

NAME            TYPE  BLOCKS  MODIFIED

*PROGRAMM1       BAS       1  NO DATE

BLOCKS FREE: 272    BLOCKS USED: 8
```

Genau umgekehrt wie bei LOCK wird durch den Befehl „UNLOCK PROGRAMM1“ der Schutz wieder aufgehoben und das File kann gelöscht oder überschrieben werden.

Durch den Befehl „RENAME“ kann der Name eines Files (muß kein BASIC-Programm sein) geändert werden. Das geschieht beispielsweise wie folgt:

RENAME PROGRAMM1,PROG1

Als Filenamen können nicht irgendwelche Zeichenkombinationen verwendet werden. Unter ProDOS ist der Filename auf 15 Zeichen begrenzt und darf nur Buchstaben, Zahlen und Punkte enthalten. Zusätzlich muß er auf jeden Fall mit einem Buchstaben beginnen. Umlaute (ä,ö,ü usw.) sind nicht zulässig. Unter DOS 3.3 sind bis zu 30 Zeichen lange Filenamen erlaubt, darunter auch Leerzeichen oder nichtdruckende Zeichen.

Ein ungeschütztes Programm kann mit dem Befehl „DELETE Filename“ gelöscht werden. Genau wie bei SAVE und LOAD sollte man damit jedoch vorsichtig sein, um unbeabsichtigte Katastrophen zu vermeiden. Genau genommen wird das Programm unter DOS 3.3 nicht gelöscht, sondern lediglich aus dem Inhaltsverzeichnis auf der Diskette entfernt. Mit gewissen Tricks läßt sich ein solches Programm eventuell wieder herstellen. Unter ProDOS wird das File wirklich von der Diskette gelöscht, ein „UNDELETE“ ist hier nicht möglich.

Da das Programm nun unter dem Namen PROG1 auf der Diskette gespeichert ist, können wir mit dem Programm im Computer experimentieren, ohne fürchten zu müssen, daß die Originalversion verloren geht.

Zunächst können wir den Befehl „NEW“ testen:

NEW
LIST

ergibt keine Antwort (außer dem Cursor), denn das Programm ist aus dem Memory (Speicher) des Computers gelöscht.

Gibt man ein neues Programm ein z.B.

```
10 HOME
20 PRINT "Hallo hier ist Programm 2"
30 LIST
40 PRINT "Dies ist mein eigenes Listing"
```

und startet es durch RUN, so löscht es den Bildschirm (Zeile 10) und stellt sich vor.

In diesem Programm wird der Befehl „HOME" verwendet; HOME löscht den Bildschirm sowohl bei 80-Zeichen Darstellung als auch bei 40-Zeichen pro Zeile und positioniert den Cursor oben links. Stehen auf dem Bildschirm zu viele Informationen oder solche, die nicht mehr gebraucht werden oder verwirren, so läßt sich der Bildschirm mit dem Befehl HOME löschen.

Das Programm verwendet keine Sprungbefehle (zur Änderung der Ablaufreihenfolge). Es läuft deshalb durch jede Zeile und hört nach der letzten Zeile einfach auf. Es gibt aber Programme, die nicht durch alle Zeilennummern laufen sollen. Um die Ausführung des Programms zu beenden, dient der Befehl „END". Gibt man z.B. ein:

```
35 END
RUN
```

so erscheint die letzte Information (Dies ist mein eigenes Listing) aus Zeile 40 nicht mehr.

Um Programme leichter ändern zu können, was bei der Progammentwicklung ja häufig vorkommt, bietet der Apple einige Hilfsmittel, die im nächsten Abschnitt erklärt werden.

3.4 Der Editor, Programme ändern

Befehle: **Cursortasten, ESCAPE, DEL**

Bisher haben wir Programme geändert, indem wir neue Zeilen eingefügt haben. Die Mehrzahl aller Änderungen betrifft aber Umstellungen in den Zeilen selbst. In diesem Abschnitt werden deshalb die vom Apple //c angebotenen Hilfen für die Programmzeilenänderungen, auch „Der Editor" genannt, beschrieben.

Auf der Tastatur in der untersten Reihe rechts sind die vier Pfeiltasten angeordnet, die den Cursor bewegen. Ist der Computer in der Betriebsart Einzelanweisungen (also APPLESOFT, ohne daß ein Programm ausgeführt wird), so funktionieren nur drei dieser Tasten: Die Pfeile rechts, links und abwärts. Der Aufwärtspfeil arbeitet nur im Zusammenhang mit Escape (siehe unten).

Wird ein Befehl eingegeben, bei dem die Abwärtspfeiltaste betätigt wurde, so erfolgt bei der Ausführung die Fehlermeldung „SYNTAX ERROR".

Wenn Sie die ESCAPE-Taste (Esc) links oben auf der Tastatur drücken, so sehen Sie, wie sich der blinkende Cursor ändert; er zeigt nun die Form eines invertierten Pluszeichens (+). Durch wiederholtes Drücken der Esc Taste kann man zwischen normalem und invertiertem Cursor wechseln. Der invertierte Cursor kann durch die Pfeiltasten in alle vier Richtungen beliebig bewegt werden.

Der Editor hat nun folgende Eigenschaft:
Wird der normale Cursor mit den horizontalen Pfeiltasten bewegt, so liest der Computer den dabei überstrichenen Text.
Wird der invertierte Cursor bewegt, egal in welche Richtung, so wird der dabei überstrichene Text nicht gelesen.
Diese Eigenschaft erlaubt uns nun, Programmzeilen zu ändern oder zu ergänzen, ohne diese erneut eingeben zu müssen. Wollen wir beispielsweise die in dem oben beschriebenen Programm vorhandene Zeile 30 ergänzen, um bei der Ausführung des Programms nach dem Listing weitere Freizeilen zu erzeugen, so geben wir ein:

LIST 30

und erhalten

```
30 LIST
```

, die Programmzeile 30, und den Cursor zwei Zeilen tiefer.
Wir drücken nun Esc, erhalten damit den invertierten Cursor (+), fahren mit dem Aufwärtspfeil (rechts unten auf der Tastatur) zwei Zeilen nach oben, bis der Cursor auf der 3 (von der Zeilennummer 30) steht. Anschließend drücken wir erneut Esc. Der jetzt wieder normale Cursor wird mit der Rechtspfeiltaste über die Zeile bewegt. Dabei liest der Computer die Zeile 30 neu ein. Ist die ganze Zeile 30 gelesen, so drücken wir nicht RETURN, sondern geben einfach den weiteren Text ein (hier wieder die Abkürzung „?" für PRINT)

```
30 LIST  : ? : ?
   ↑        ↑
eingelesener neu eingegebener
Teil         Teil
```

und drücken RETURN. Zur Kontrolle dient wieder der Befehl LIST:

LIST 30

ergibt die Antwort

```
30 LIST : PRINT : PRINT
```

Auf diese Weise haben wir die schon existierende Zeile 30 um weitere Befehle ergänzt. Ganz analog geschieht das Verbessern von Tippfehlern; dabei werden die richtig geschriebenen Partien der Zeile mit Hilfe des Cursors eingelesen und falsche Teile einfach überschrieben.

Will man zwischen schon existierende Befehle einer Zeile einen weiteren Befehl einfügen, so liest man die betreffende Zeile wie oben beschrieben, bis ein Zeichen nach dem gewünschten Doppelpunkt, der zwei Befehle trennt, fährt mit Hilfe von Esc, Aufwärtspfeil und wieder Esc eine Zeile höher, und schreibt dort den neuen Befehl. Anschließend drückt man wieder Esc, fährt zu dem erwähnten Doppelpunkt zurück, drückt wieder Esc, um den normalen Cursor zu erhalten, und liest den Rest der Zeile ein.

Ganz allgemein funktioniert das Ganze also so:
Der normale Cursor liest alle Zeichen auf dem Bildschirm, über die er mit den Pfeiltasten bewegt wird, in den Computer ein (natürlich maximal 239 Zeichen). Nach dem Return wird die so eingelesene Zeichenfolge direkt ausgeführt, wenn sie nicht mit einer Zahl (Zeilennummer) beginnt. Andernfalls wird sie als Programmzeile betrachtet. Es ist also so, als ob die Zeichenfolge über die Tastatur eingegeben worden wäre.

Der invertierte Cursor, der durch Drücken der ESCAPE-Taste (Esc) erzeugt wird, kann mit den vier Pfeiltasten beliebig bewegt werden, ohne sich irgendwie um den überstrichenen Text zu „kümmern".

Ist man sich über diese beiden Cursoren im Klaren, so kann man mit den Pfeiltasten und geeignetem Einsatz der Esc-Taste, um zwischen den beiden Cursoren zu wechseln, beliebige einfache Umwandlungen vornehmen.

Ebenso können neue Zeilen aus schon bestehenden Zeilen erzeugt werden, indem man die alte Zeilennummer ändert und den Text der alten Zeile mit dem normalen Cursor einliest. Die alte Zeile bleibt bei dieser Prozedur natürlich erhalten.

```
]100 PRINT "Urspruengliche Zeile"
]
```

Den normalen Cursor ▮ mit [esc] invertieren ▯

```
]100 PRINT "Urspruengliche Zeile"
]▯
```

Mit der [↑] Taste zur 1 (von Zeilennr. 100)

```
]▯00 PRINT "Urspruengliche Zeile"
```

Mit [esc] den Cursor wieder normalisieren, und bestehende Zeile soweit gewuenscht lesen.

```
]100 PRINT "▮rspruengliche Zeile"
```

Neuen Text eingeben:

```
]100 PRINT "Neue▮uengliche Zeile"
```

Cursor mit [esc] invertieren ▯, und ueber den nicht brauchbaren Text bewegen.

```
]100 PRINT "Neue▯uengliche Zeile"
]100 PRINT "Neueruengliche▯Zeile"
```

Die Buchstaben"ruengliche" wurden dabei nicht kopiert. Den Cursor wieder mit [esc] normalisieren, und den Rest einlesen.

```
]100 PRINT "Neueruengliche Zeile"▮
```

Mit RETURN abschliessen, und das Ergebnis mit LIST kontrollieren.

```
]LIST 100
]100 PRINT "Neue Zeile"
```

Sollen Zeichen gelöscht werden, so überschreibt man sie am besten mit der Leertaste. Zuviel eingebene Leerzeichen stören nicht, sie werden vom Computer herausgefiltert, außer sie stehen zwischen Anführungszeichen.

Die beschriebenen Eigenschaften des normalen und des invertierten (Esc) Cursors funktionieren auch für Einzelanweisungen.

Will man ganze Zeilen löschen, so gibt man einfach nur die betreffende Zeilennummer neu ein. Damit wird eine neue Zeile (die ohne Inhalt ist) erzeugt, und damit die alte gelöscht. Die neue Zeile ohne Inhalt wird von selbst beseitigt.

Durch den Befehl DEL können mehrere Zeilen auf einmal gelöscht werden:

DEL 20,30	löscht alle Zeilen zwischen 20 und 30 (inklusive)
DEL ,30	löscht alle Zeilen vom Anfang bis Zeile 30 (inklusive)
DEL 30,	löscht allen Zeilen ab 30 bis zum Ende des Programms

Im Handel werden komfortable BASIC-Editoren für den Apple angeboten, welche zusätzliche Möglichkeiten bieten. Mit normalen Textverarbeitungsprogrammen können ebenfalls BASIC-Programme geschrieben und geändert werden. Die dafür notwendige Prozedur ist im Abschnitt „Programmierhilfen" beschrieben.

3.5 Variable, Daten, Texte, Zahlen

Befehle: **DIM, FRE(0)**

Der Begriff Variable hat eine zentrale Stellung bei der Programmierung. Eine Variable ist ein Name für einen numerischen Wert oder eine Zeichenkette. Das soll folgendes Beispiel verdeutlichen:
Die Berechnung von 12 mal 12 kann direkt geschehen durch:

```
PRINT 12 * 12
144
```

Sie kann auch über eine Wertzuweisung stattfinden:

```
A = 12 * 12
PRINT A
144
```

Der Unterschied zwischen beiden Eingaben ist folgender:
Während im ersten Fall der Wert (12 mal 12) berechnet und sofort ausgegeben wird, definieren wir im zweiten Fall eine Variable A, der der Wert (12 * 12) zugewiesen wird. Anschließend wird die Variable A ausgegeben. Die Definition der Variable geschieht bei BASIC automatisch bei der erstmaligen Erwähnung, der Benutzer muß sich also in den meisten Fällen nicht darum kümmern. Ausnahmen werden unten erklärt.

Soll der Wert 144 weiter verarbeitet werden (z.B. soll er durch 3 dividiert werden), so muß im ersten Fall die gesamte Rechnung wiederholt werden:

```
PRINT 12 * 12 / 3
48
```

Im zweiten Fall, bei der Benutzung einer Variable A, genügt:

```
PRINT A / 3
48
```

Bei der (automatischen) Definition einer Variablen (hier A) und der Wertzuweisung (A = Wert) bleibt also das Ergebnis unserer Berechnung im Computer gespeichert.
In allen Fällen, wo wir uns (bzw. das Programm) Informationen irgendwelcher Art merken müssen, geschieht das über die Zuweisung zu einer Variablen.
Dieser Fall tritt im allgemeinen sehr häufig auf, zum Beispiel dann, wenn die weitere Verarbeitung der Variablen von ihrem Wert abhängt, oder wenn mehrere Werte gleichzeitig gespeichert werden müssen.

Soviel zum Sinn der Variablen, nun zur detaillierten Erklärung. In APPLESOFT werden drei verschiedene Typen von Variablen unterschieden:

1. Gleitkomma-Variable, deren Name nur mit Buchstaben und Zahlen gebildet wird (A, A1, PQ, AD22, PRODUKT7).

2. Festkomma- oder Integer-Variable, deren Name mit Buchstaben und Zahlen gebildet wird und mit dem Prozentzeichen (%) endet (A%, A1%, SUM%, X212%)

3. Text- oder String-Variable, deren Name ebenso nur aus Buchstaben und Zahlen gebildet wird und mit dem Dollarzeichen ($) endet (A$, A1$, NAME$).

Die Unterscheidung zwischen Zahlenvariablen und Textvariablen (auch Zeichenketten genannt) leuchtet ein: mit Zahlenvariablen sind Berechnungen möglich, mit Textvariablen nicht.
Zahlenvariablen sind aber wie erwähnt noch weiter unterteilt in Integer- und Gleitkommavariable. Der Speicherbedarf von Integervariablen und Gleitkommavariablen ist gleich, jeweils 5 Byte pro Zahl (außer dimensionierte Variablen, siehe DIM). Die Unterschiede werden jeweils in einem eigenen Abschnitt besprochen.
Alle Variablennamen müssen mit einem Buchstaben beginnen, danach können Zahlen und Buchstaben in beliebiger Reihenfolge verwendet werden. Die Variablenbezeichnung kann bis zu 238 Zeichen lang sein. APPLESOFT-BASIC

unterscheidet aber nur die ersten beiden Zeichen eines Variablennamens. Die Anweisungen

```
SUM1% = 1000
SUM2% = 2000
PRINT SUM1%
```

haben die Antwort

```
2000
```

zur Folge, da das dritte und vierte Zeichen (und eventuell alle weiteren) nicht zur Unterscheidung herangezogen, wohl aber ausgedruckt werden.
Die Endzeichen % für Integer- und $ für Textvariable werden trotzdem berücksichtigt; die Anweisungen

```
NUMMER% = 999
NUMMER = 1000
PRINT NUMMER%
```

ergeben

```
999
```

3.5.1 Numerische Variablen

Der geringere Speicherbedarf der dimensionierten Integervariablen (siehe auch 3.5.3) rührt von dem eingeschränkten Wertebereich her. Integer-Variable können nur ganzzahlige Werte zwischen -32767 und +32767 (inklusive) annehmen. (Entspricht der 2 Byte mal (8 Bit) Auflösung: (2 hoch 8) hoch 2 = 65356). Andere Werte werden nicht akzeptiert und haben einen „ILLEGAL QUANTITY ERROR“ zur Folge.

Der zulässige Wertebereich für Gleitkommavariable ist nicht plus/minus unendlich, sondern +/- 10 hoch 38. Dabei ist zu unterscheiden zwischen der Genauigkeit der Darstellung und dem tatsächlichen Wert der Variable.
Gleitkommazahlen werden in APPLESOFT mit maximal 9 Stellen dargestellt, aber genauer gerechnet. Die Darstellung wird jeweils gerundet. Die Eingabe von:

PRINT 1/3

hat den Dezimalbruch

.333333333

als Ergebnis, der auf maximal 9 Stellen genau angegeben wird. Bei dieser Darstellung läßt der Apple die Zahl 0 vor dem Dezimalpunkt weg. Computer verwenden den Dezimalpunkt statt des im Deutschen verwendeten Kommas. Das Komma dagegen wird manchmal zur Trennung von jeweils 3 Stellen vor dem Dezimalpunkt verwendet (Tausenderstellen).

Werte, die mehr als 9 Ziffern vor dem Komma (dem Dezimalpunkt) haben oder absolut kleiner als 0.01 sind, erhalten eine andere Schreibweise:

```
PRINT .000000000000000000000045
4.5 E-24

PRINT 1234567890
1.23456789 E+09
```

Die Darstellung mit dem Buchstaben E (Exponentenschreibweise) bedeutet: 1. Zahl * 10 hoch Zahl hinter „E" (z.B. 4.5 * 10↑(-24)); diese Schreibweise ist dann sinnvoll, wenn der Wert der Zahl sehr groß ist oder so nahe bei Null liegt, daß die normale Darstellung sehr lang würde.

Werden Gleitkommazahlen in Integerzahlen umgewandelt, beispielsweise durch

```
A=12345.678
A%=A
PRINT A%
12345
```

,so werden die Ziffern nach dem Komma weggelassen, also nicht gerundet.

3.5.2 Textvariable

Textvariable oder Strings sind eine Folge von Zeichen, deren Name mit dem Dollarzeichen ($) aufhört. Die Zeichen des Strings (nicht des Namens) sind

nicht auf Buchstaben und Zahlen begrenzt, sondern es können auch andere Zeichen wie Punkte, Kommas usw. gebraucht werden. Außerdem können sogenannte nichtdruckende Zeichen verwendet werden, die zwar vorhanden sind, aber den Computer nicht zur Darstellung auf dem Bildschirm veranlassen. Ein solches Zeichen ist Control-G, auch „Bell" (Glocke) genannt.
Zur Eingabe des Strings werden die Anführungsstriche (") verwendet:

A$ = "Hallo, ich bin ein String !"
PRINT A$
Hallo, ich bin ein String !

Die Eingabe von nichtdruckenden Zeichen erfolgt genauso, nur werden diese nicht dargestellt. Zwischen den beiden Anführungszeichen (") kann man z.B. gleichzeitig auf die Tasten Control und G drücken. Dies ändert am Bildschirminhalt nichts, aber Control-G ist dann im String enthalten. Läßt man diesen String mit PRINT ausgeben, so ertönt ein Signal aus dem Lautsprecher (vorausgesetzt die Lautstärke ist nicht abgestellt), daher auch der Name Bell.

```
10 PRINT  "My name is bell"
              ↑
    Hier Control-G drücken
```

RUN

My name is bell (mit Ton aus dem Lautsprecher)

Nichtdruckende Zeichen dienen vor allem der Steuerung von externen Geräten, z.B. Druckern. Mit dem Befehl CHR$ können nichtdruckende Zeichen erzeugt werden, die auf der Tastatur nicht vorhanden sind; dies ist im Abschnitt „Ein- und Ausgabeprogramme" erläutert.

3.5.3 Matrizen (Arrays, Felder)

Eine Variable in einem Programm oder bei Einzelanweisungen benötigt Speicherplatz. Dieser Speicherplatz wird ihr gewöhnlich automatisch zugeordnet. Variablen können jedoch auch in Form von Feldern, sogenannten Matrizen oder Arrays, verwendet werden. In diesem Fall haben sie hinter ihrem Namen in Klammern Angaben, wo sie im Feld zu finden sind. Das kann man sich etwa so vorstellen:

```
A(1) A(2) A(3) A(4) A(5) A(6) A(7) A(8) A(9)
```

oder

```
A(1,1)    A(1,2)    A(1,3)    A(1,4)    A(1,5)
A(2,1)    A(2,2)    A(2,3)    A(2,4)    A(2,4)
A(3,1)    A(3,2)    A(3,3)    A(3,4)    A(3,4)
A(4,1)    A(4,2)    A(4,3)    A(4,4)    A(4,5)
```

Das erste Feld heißt eindimensional (auch Vektor genannt), das zweite zweidimensional (hier eine 4 mal 5 Matrix). Ein dreidimensionales Feld wäre nur räumlich darstellbar; 4- und mehrdimensionale Felder kann der Mensch sich nicht vorstellen, trotzdem sind sie möglich.

Diese Arrays oder Felder brauchen sehr viel Speicherplatz; der zweite Array benötigt z.B. 5 x 5 x 4 = 100 Byte. Aus diesem Grund sollte man auch große Arrays möglichst auf Integerzahlen (Ganze Zahlen zwischen -32767 und +32767) beschränken, diese benötigen in der Arrayform nur 40% des Platzes der Gleitkommazahlen (2 statt 5 Byte pro Zahl).
Für die Ausdehnung der Felder in jede Richtung gibt es keine festgelegten Höchstgrenzen. Sie sind lediglich durch den benötigten Speicherplatz für das gesamte Feld begrenzt.

Kleine Felder, wie die oben dargestellten, können ohne vorherige Definition einfach verwendet werden. Übersteigt jedoch die Ausdehnung des Feldes in einer Richtung die Zahl 10, z.B. A(15) oder B%(3,16,4), so muß der dafür benötigte Platz im Speicher des Computers reserviert werden. Das geschieht mit dem Befehl DIM:

DIM S1%(20,45)

erzeugt ein Feld von 21 x 46 = 966 Integerzahlen, von S1%(0,0) bis S1%(20,45). Nach dem DIM-Befehl wird der Variablenname angegeben, in Klammern gefolgt von der Angabe der Ausdehnung des Arrays in alle vorgesehenen Richtungen.

Jeder Array darf nur einmal definiert werden. Man kann also Arrays im Bedarfsfall nicht nachträglich erweitern, weil sonst die Fehlermeldung „REDIM'D ARRAY ERROR“ erscheint.

Werden Arrayvariable verwendet, deren Indices (die Angaben über den Ort der Variable im Feld) außerhalb des definierten Arrays liegen, so erfolgt ein „BAD SUBSCRIPT ERROR“.

3.5.4 Freier Speicherplatz, FRE(0)

Der Apple //c hat im Normalfall etwa 36 kB freien Speicherplatz (Memory) zum Programmieren. Weiterhin sind nochmal ca. 60 kB „in Reserve" vorhanden, deren Benutzung allerdings etwas komplizierter ist. Normalerweise reichen die ersten 36 kB Speicherplatz für die Programme. Trotzdem kann es in zwei Fällen zu Engpässen kommen:

1. Bei Verwendung großer Datenmengen, insbesondere bei Arrays.
2. Bei Grafikprogrammen.

Im zweiten Fall können die Befehle HIMEM: und LOMEM: nützlich sein, die später erklärt werden. Im hier wichtigeren Fall, bei großen Datenmengen, ist der Befehl FRE(0) von Bedeutung:

```
PRINT FRE(0)
24523
```

FRE(0) liefert die Anzahl noch freier, ungebrauchter Speicherplätze in Bytes, hier ca. 24 kB. Diese Zahl ist natürlich je nach vorhandenem Programm und Daten jeweils anders. Wird eine Zahl kleiner als Null angezeigt, so müssen Sie 65536 addieren, um den tatsächlichen freien Speicherplatz zu erhalten. Dies hängt mit der internen Nummerierung des RAM Speichers zusammen. Gelegentlich kann es eine merkliche Zeit dauern, bis das Ergebnis erscheint, da bei der Feststellung der freien Plätze vom Rechner gleichzeitig der Speicher „aufgeräumt" wird, indem nicht mehr benutzte Variablen gelöscht werden.
Bei der Eingabe von Daten oder Programmen läßt sich leicht kontrollieren, wie der freie Speicherbereich kleiner wird:

```
PRINT FRE(0)
28501

DIM A%(999)
PRINT FRE(0)
26494            (braucht 1000 * 2 + 7 Byte)

DIM A(999)
PRINT FRE(0)
21487            (braucht 1000 * 5 + 7 Byte)
```

```
                Z = 12.4
                PRINT FRE(0)
21480           (braucht 7 Byte)

Z$ = "Ich habe 20 Zeichen!"
PRINT FRE(0)
21453           (braucht 20 + 7 Byte)
```

Bei der Dimensionierung eines (eindimensionalen) Feldes von 1000 Integerzahlen DIM A%(999) (gibt 1000 Zahlen, die von 0 bis 999 durchnumeriert sind), sinkt der freie Speicherplatz um 1000 mal 2 Byte pro Integerzahl plus 7 Byte, also um 2007 Byte. Die 7 Byte zusätzlich zur eigentlichen Datenmenge dienen zur Markierung des Felds, besser gesagt zur Markierung des vom Feld belegten Speicherbereichs.
Bei der Dimensionierung von 1000 Gleitkommazahlen, die jeweils 5 Byte beanspruchen, sinkt der noch freie Bereich entsprechend um 5007 Byte. Die Verwendung der Variablen Z benötigt nochmals 7 Byte, und die Variable Z$ braucht 27 Byte; nämlich zunächst 7 Byte, um den von der Variablen belegten Platz zu kennzeichnen, sowie jeweils ein Byte für jedes Zeichen, also 7 + 20 = 27 Byte.

Werden in einem Programm große Datenmengen gebraucht, so sollte man vor der Speicherplatzreservierung (durch den Befehl DIM) die Größe des noch vorhandenen Speicherplatzes mit FRE(0) bestimmen, um im Programm einen „OUT OF MEMORY ERROR“ zu vermeiden.

3.6 Die Rechenoperationen

Befehle: **(), +, -, *, /, =, <, >, MOD, NOT, AND, OR, (IF ... THEN)**

Die Rechenoperationen umfassen die arithmetischen Operationen wie + oder *, vergleichende Operationen wie > oder =, sowie Bool'sche Operationen wie AND. Daneben lassen sich mit dem Apple //c Texte „addieren“.

Die Rechenoperationen dienen dazu, Variablen und Zahlen zu verknüpfen. Die verschiedenen Operationen sind in einer Rangordnung festgelegt, die aus der Mathematik bekannt ist.

Bei Ausdrücken, in denen mehr als eine Operation vorkommt, z.B.

A = 2.4 + (12.57 / B) ↑ 2

werden die Operationen nach der in der Rangordnung festgelegten Reihenfolge ausgeführt.

Den höchsten Rang haben die Klammern. Sie werden als erste ausgeführt, bei geschachtelten Klammern von innen nach außen. Die Eingaben

```
A = 2 + 3 * 4
B = ( 2 + 3 ) * 4
PRINT "A=";A,"B=";B
```

ergeben

```
A=14          B=20
```

Bei der Berechnung von A wird zunächst die Multiplikation (3 * 4) durchgeführt, anschließend addiert. Umgekehrt bei B, wo zunächst addiert und anschließend multipliziert wird. Die Änderung der Reihenfolge ist auf die Klammer zurückzuführen, die einen Vorrang vor der Multiplikation hat. Die Addition geschieht bei der Berechnung von A erst nach der Multiplikation, gefolgt von der Wertzuweisung =.
Durch solche Tests läßt sich die Rangordnung der Operationen feststellen. Sie ist in der folgenden Tabelle festgehalten.

Rangordnung der Ausführung		Operation
1	()	Klammern
2	-	Vorzeichen
3	↑	Potenz
4	*	Multiplikation
4	/	Division
5	+	Addition
5	-	Subtraktion
6	=	Gleichheit
6	<	kleiner

Rangordnung der Ausführung		Operation
6	>	größer
6	><	ungleich (><)
6	<=	kleiner gleich (=<)
6	>=	größer gleich (=>)
7	NOT	Negation
7	AND	logisches Und
7	OR	logisches Oder

Operationen gleichen Ranges werden in der Reihenfolge ihrer Darstellung in der Berechnungsformel durchgeführt, solange nicht durch Klammern eingegriffen wird.
Anschließend an die dargestellten Operationen werden Vergleiche und Wertzuweisungen ausgeführt; zuletzt schließlich Bool'sche Operationen. Vergleiche, Wertzuweisungen und Bool'sche Operationen sind unten erklärt.

Texte können durch Addition zusammengefügt werden. Dabei darf die Gesamtlänge des Textes nicht größer als 255 werden. Beispiel:

```
A$ = "zusammengesetzter"
B$ = "Dies ist ein " + A$ + " Satz"
PRINT B$
Dies ist ein zusammengesetzter Satz
```

3.6.1 Vergleichende Ausdrücke

Ergebnisse der vergleichenden Ausdrücke sind immer die Zahlen 0 oder 1, je nach dem, ob der Vergleich zutrifft (wahre Aussage, Ergebnis = 1) oder nicht (falsche Aussage, Ergebnis = 0). Die vergleichenden Ausdrücke stehen häufig im Zusammenhang mit der Befehlskombination „IF ... THEN", bei der aufgrund eines Vergleichs der Programmablauf beeinflußt werden kann:

```
IF 33 > 32 THEN ? "Richtig"
Richtig
IF 20 < = 5 * 4 THEN ? "auch der zweite Vergleich stimmt"
auch der zweite Vergleich stimmt
IF 20 < 5 * 4 THEN ? "Falsch"
```

Der letzte Vergleich führt zu keiner Antwort, da der Vergleich 20 < 20 (= 5 * 4) negativ ausfällt, das Ergebnis also 0 ist.

Die Kombination „IF ... THEN“ hat jeweils die Form:

IF Vergleich THEN Befehl

,wobei der Befehl nur dann ausgeführt wird, wenn der Vergleich stimmt. Ansonsten wird das Programm mit der nächstfolgenden Zeilennummer fortgetzt.

Statt Zahlen können auch Strings verglichen werden:

```
IF "AA" < "AB" THEN ? "AA ist kleiner als AB"
AA ist kleiner als AB

A$ = "EISEN" : B$ = "BAHN"
IF A$ + B$ = "EISENBAHN" THEN ? "Richtig"
Richtig

IF A$ + B$ >< B$ + A$ THEN ? "A$ ungleich B$ "
A$ ungleich B$
```

Dabei werden die Länge der Texte und der Inhalt verglichen. Grundsätzlich ist der längere Text, also der der mehr Zeichen besitzt, der größere.
Der Inhaltsvergleich basiert wieder auf einem Zahlenvergleich. Allen Zeichen sind Zahlen zugeordnet, die sogenannten ASCII-Zahlen (siehe Anhang); die großen Buchstaben beispielsweise haben die Zahlen A = 65, B = 66, C = 67, etc. bis Z = 90. Der Computer vergleicht dann nur die ASCII-Zahlen, und zwar jeweils die ersten beider Strings, dann die zweiten, usw. Der String, der als erster, vom Stringanfang aus gesehen, eine höhere ASCII-Zahl hat, ist der größere. Das Wort „HALLI“ ist kleiner als das Wort „HALLO“.

3.6.2 Bool'sche Operationen

Analog zu den vergleichenden Ausdrücken haben Bool'sche Operationen als Ergebnis nur 0 oder 1. Sie werden deshalb auch für die „IF ... THEN“ Kombination als Vergleich benutzt. Die drei in APPLESOFT vorhandenen Bool'schen Operatoren sind: AND, OR und NOT.

Die Bedeutung des AND (= UND) geht aus folgender Zeile hervor:

IF 2 = 2 AND 3 = 4 THEN ? "Alles OK"

Es wird keine Antwort gedruckt. Das Wort AND verknüpft beide Bedingungen für die IF ... THEN Kombination (2 = 2, sowie 3 = 4) und gibt den Wert „wahr", also 1, an das Wort IF nur dann weiter, wenn beide Bedingungen erfüllt sind. In allen anderen Fällen wird der Wert 0 weitergegeben. Das Wort AND kann auch mehrmals hintereinander verwendet werden:

Bedingung1 AND Bedingung2 AND Bedingung3

ergeben nur dann den Wert 1 (wahr), wenn alle drei Bedingungen erfüllt sind.

Ähnlich ist die Bedeutung des Wortes OR (= ODER).Bei diesem Wort wird der Wert 1 (wahr) dann weitergegeben, wenn wenigstens eine der Bedingungen, die durch OR verknüpft sind, zutrifft:

IF 2 = 2 OR 3 = 4 THEN ? "Alles OK"
Alles OK

Werden mehrere AND und OR Befehle kombiniert, so werden sie der Reihenfolge nach abgearbeitet, wenn nicht durch das Setzen von Klammern ein anderer Ablauf erzwungen wird.

IF 2 = 2 OR 3 = 4 AND 5 = 5 THEN ? "Alles OK"

ergibt keine Antwort, hingegen

IF 2 = 2 OR (3 = 4 AND 5 = 5) THEN ? "Alles OK"
Alles OK

Das Wort NOT (= NICHT) negiert lediglich die von OR, AND, <, <=, =, <=, <>, >< oder < ausgegebenen Werte 0 oder 1; d.h. es macht aus einer 0 eine 1 und umgekehrt aus einer 1 eine 0. Das Wort NOT kann mit den vergleichenden und Bool'schen Operanden beliebig kombiniert werden:

IF 2 = 3 THEN PRINT "2=3"

```
IF NOT 2 = 3 THEN PRINT "2 ist nicht 3"
2 ist nicht 3
```

Mit Ausnahme der Textoperationen lassen sich alle anderen Verknüpfungen von Zahlen beliebig mischen. So können auch die von vergleichenden oder Bool'schen Operationen gelieferten Ergebnisse 0 oder 1 zur Weiterverarbeitung verwendet werden:

```
A = 20 : B = 36
? ( A < 21 ) * 100 + ( B = 36 ) * 10 + (( B < 50 ) AND ( A = 20 )

110
```

Die dargestellte Berechnung enthält drei Vergleiche, die jeweils in Klammern geschrieben sind: (A < 21), (B = 36) und ((B > 50) AND (A = 20)). Die beiden ersten sind wahr, haben daher den Wert 1. Die letzte ist untergliedert; ihr Wert ist 0, da B kleiner als 50 ist, also das Ergebnis 0 liefert. Der zweite Teil des letzten Vergleichs ist zwar eins, der Gesamtvergleich wird aber gleich 0 wegen der Verknüpfung des zweiten Teils durch AND mit dem ersten Teilvergleich (Ergebnis 0).
Das Gesamtergebnis setzt sich dann aus den Teilbeträgen

1 * 100 + 1 * 10 + 0 = 100 + 10 + 0 = 110

zusammen.

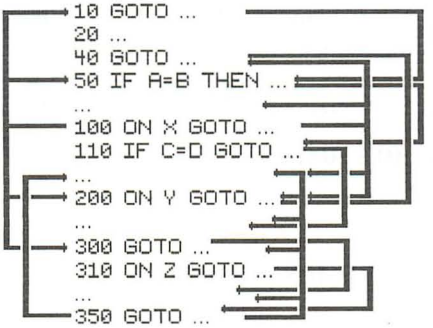
3.7 Die Programmverzweigungen

Befehle: **GOTO, GOSUB, RETURN, POP, ON ... GOTO, ON ... GOSUB, IF ... THEN, FOR ... NEXT**

Die bisher beschriebenen Befehle erlauben eine Vielzahl von Berechnungen und Datenmanipulationen. Die dabei angewendeten Programme liefen alle kontinuierlich ab. Dies ist immer dann der Fall, wenn eine begrenzte Reihe von Operationen von der ersten bis zur letzten ausgeführt werden soll. In den meisten Fällen ist es aber zweckmäßig, die Reihenfolge des Ablaufs zu ändern. In diesem Kapitel werden die Möglichkeiten zur Unterbrechung dieses Ablaufs erläutert.

3.7.1 Einfache Sprünge, GOTO

Das bekannteste Wort der Sprache BASIC ist vermutlich das Wort GOTO. Das Wort ist selbsterklärend, insbesondere in der Form GO TO. GO TO bedeutet GEH ZU und veranlaßt einen Sprung zu der Zeile, deren Nummer nach GOTO steht. Leider hat die allzuhäufige Verwendung des Wortes zum schlechten Image der Sprache BASIC geführt, Programme in Spagettistruktur zu provozieren.



Programm mit Spagettistruktur.

Das Beispiel zeigt die Form eines Programms mit Spagettistruktur. Je länger das Programm ist, umso schwieriger ist es, den Sinn und die Logik eines solchen Programms zu erfassen, selbst für denjenigen, der es geschrieben hat. In der Sprache FORTRAN, wo das Wort GOTO auch vorkommt, wird es allgemein viel weniger gebraucht.

Die Qualität eines Programmes ist umgekehrt proportional zur Summe der GOTOs.

Diese etwas lange Betrachtung der GOTO-Philosophie soll Sie vor dem warnen, was Sie sich selbst mit schlechten Programmen antun können, wenn Sie zu viele GOTOs am falschen Platz verwenden.

```
10 PRINT  : PRINT  "HALLO"
20 GOTO 40
30 PRINT  "NOCHMALS HALLO"
40 PRINT  "ENDE"
RUN
```

HALLO
ENDE

Der Befehl „GOTO Zeilennummer“ überspringt Zeile 30 des Programms. Stattdessen wird mit Zeile „Zeilennummer“ fortgefahren. Dies ist der einfachste Fall von GOTO. Die einzige Bedingung für ein erfolgreiches GOTO ist das Vorhandensein der Sprungadresse (hier Zeile 40). GOTO arbeitet hier anders als LIST, welches die nächsthöhere Zeile listet, wenn die aufgerufene Zeile nicht existiert. Ist die nach GOTO angegebene Zeilennummer nicht vorhanden, so erscheint die Fehlermeldung „UNDEFINED STATEMENT ERROR“, und das Programm wird gestoppt.

```
10 PRINT  : PRINT  "HALLO"
20 GOTO 40
30 PRINT  "NOCHMALS HALLO"
RUN
```

HALLO
? UNDEFINED STATEMENT ERROR

Die Sprungadresse des Befehls GOTO 40 existiert nicht, daher die Fehlermeldung.
Die Zeilennummer hinter dem GOTO-Befehl kann nicht berechnet werden. Eine Möglichkeit, trotzdem zu berechneten Sprungadressen zu verzweigen, bietet die Befehlskombination ON ... GOTO.

```
10 ON A% GOTO 100, 110, 180, 1330, 50, 60
```

Hier ist A% eine vorher berechnete ganze Zahl zwischen 1 und 32767. Wird eine Dezimalzahl berechnet, so werden die Stellen nach dem Komma abgeschnitten, die Zahl also abgerundet. Hier wird mit A% = 1 nach Zeile 100 verzweigt, bei A% = 2 nach 110, usw. Bei A% = 6 wird nach 60 verzweigt. Die Zeilennummern können im Bedarfsfall leicht geändert werden.

Es ist von Vorteil, wenn nach dem ON ... GOTO Befehl ein weiterer Befehl steht; z.B. eine GOTO-Anweisung. Diese wird dann ausgeführt, wenn für den Wert zwischen ON und GOTO keine Sprungadresse angegeben ist.

```
10 ON B% GOTO 100, 200, 150, 300 : GOTO 20
```

GOTO 20 wird ausgeführt, wenn B% größer als 4 oder kleiner als 1 ist.

Sehr häufig steht der Befehl GOTO in Verbindung mit der IF ... THEN Kombination. Diese ist weiter unten im Abschnitt erklärt.

3.7.2 Unterprogramme, GOSUB

Der Befehl „GOSUB Zeilennummer" arbeitet sehr ähnlich zum GOTO Befehl. Auch er verzweigt zur Zeile „Zeilennummer". Der Befehl GOSUB müßte im Prinzip GOSUB ... RETURN heißen, denn GOSUB hat folgende Eigenschaft: Der APPLESOFT-Interpreter speichert die Zeilennummer, bei der ein GOSUB Befehl durchgeführt wurde. Steht im weiteren Ablauf des Programms der Rückkehrbefehl RETURN, so springt der Interpreter zu der Adresse zurück, bei der der GOSUB Befehl stand.

```
10 GOSUB 100
20 PRINT  "ENDE"
30 END
100 PRINT  : PRINT  "Unterprogramm 100"
110 RETURN
RUN
```

Unterprogramm 100
ENDE

In diesem Programm wird nach Zeile 10 zunächst Zeile 100 ausgeführt. Anschließend können beliebig viele weitere Befehle bearbeitet werden. In Zeile 110 trifft der Interpreter auf den Befehl RETURN und springt zurück zu Zeile 10. Da diese Zeile schon vollständig ausgeführt worden ist, wird mit Zeile 20 weitergemacht. Stehen in der Zeile des GOSUB-Befehls (hier Zeile 10) weitere Befehle, so werden diese nach dem durch RETURN veranlaßten Rücksprung durchgeführt.

```
10 GOSUB 100: PRINT  "Rückkehr aus Unterprogramm 100"
20 PRINT  "ENDE"
30 END
100 PRINT  : PRINT  "Unterprogramm 100"
110 RETURN
RUN
```

Unterprogramm 100
Rückkehr aus Unterprogramm 100
ENDE

Die Befehlskombination GOSUB ... RETURN ermöglicht die Strukturierung der Programme. Die Befehlsfolgen, die nach einem GOSUB bis zum nächsten

RETURN ausgeführt werden, heißen Unterprogramm, auf Englisch „subroutine". Daher kommt auch der Name GOSUB.

Die Vorteile von GOSUB liegen in der Verkürzung der Programme, da alle Programmteile, die mehrmals gebraucht werden, als Unterprogramme nur einmal geschrieben werden müssen. Gleichzeitig wird die Lesbarkeit der Programme erhöht. Der grobe Programmablauf wird schnell aus dem Listing deutlich.

Unterprogramme können in maximal 24 Ebenen geschachtelt werden wie Klammern. Das heißt, maximal 24 GOSUBs können durchgeführt werden, bevor das erste RETURN auftreten muß.

```
10 GOSUB 100
20 PRINT  "ENDE"
30 END
100 PRINT  : PRINT  "Unterprogramm 100"
105 GOSUB 200
110 RETURN
200 PRINT  : PRINT  "Unterprogramm 200"
210 PRINT  "Ich bin ein Unterprogramm innerhalb von Unterprogramm
100"
220 RETURN
RUN
```

Unterprogramm 100

Unterprogramm 200
Ich bin ein Unterprogramm innerhalb von Unterprogramm 100
ENDE

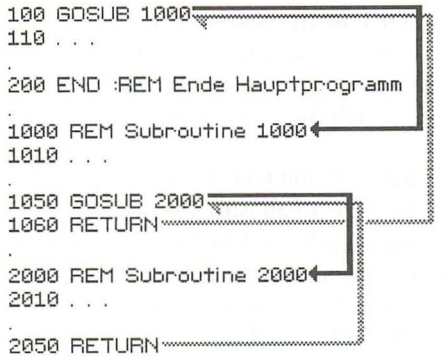
In Unterprogramm 100 wird nochmals zu Unterprogramm 200 verzweigt. Aus diesem wird dann erst wieder in Unterprogramm 100 zurückgesprungen und danach ins Hauptprogramm.
Der Begriff Schachtelung von Unterprogrammen bezieht sich auf die Reihenfolge der Ausführung, nicht auf die Darstellung im Listing.

Wenn die Anfangsadressen der Unterprogramme speziell gekennzeichnet werden, verbessert das die Lesbarkeit des Listings. Die Verdeutlichung kann mit dem REM-Befehl geschehen. REM ist die Abkürzung von Remark (Bemerkung). Der nach dem REM-Befehl bis zum Zeilenende stehende Text wird vom APPLESOFT-Interpreter ignoriert. Er dient nur dem Benutzer als Kommentar. Die Zeilennummern 100 und 200 des Programms, die die Einsprungadressen der Unterprogramme sind, können beispielweise als REM-Zeilen (Kom-

mentarzeilen) verwendet werden, in denen der Sinn des Unterprogramms kurz zusammmengefaßt ist, z.B.:

```
100 REM - SUB100, löscht Bildschirm und zeigt Hauptauswahl
```

Ein RETURN-Befehl ist nicht einem speziellen GOSUB Befehl zugeordnet, sondern der Rücksprung erfolgt immer zum letzten GOSUB-Sprung, der im Programmablauf durchgeführt wurde. Aus diesem Grund dürfen sich zwei Unterprogramme oder Subroutinen nicht überschneiden; entweder sie werden strikt nacheinander ausgeführt, oder die innere ist vollständig in der äußeren enthalten. Verstöße gegen diese Regel verursachen unkontrollierte Programmabläufe.



Geschachtelte GOSUB's

Unterprogramme können sich weder direkt selbst aufrufen noch indirekt über ein weiteres Unterprogramm. Im obigen Beispiel darf aus Subroutine 200 nicht Subroutine 100 nochmals aufgerufen werden.

Die Sprungdressen von GOSUBs können nicht berechnet werden. Analog wie bei GOTO ist die Form ON ... GOSUB möglich:

```
10 A% = 2 : ON A% GOSUB 100, 200
20 PRINT  "ENDE"
30 END
100 PRINT  : PRINT  "Unterprogramm 100"
```

```
105 GOSUB 200
110 RETURN
200 PRINT  : PRINT  "Unterprogramm 200"
210 PRINT  "Ich bin ein eigenständiges Unterprogramm"
220 RETURN
RUN
```

Unterprogramm 200
Ich bin ein eigenständiges Unterprogramm
ENDE

Hier wird Subroutine 200 direkt aus dem Hauptprogramm aufgerufen, nicht über Subroutine 100. Der Programmablauf läßt Subroutine 100 einfach aus. Es wird nur ein GOSUB und ein RETURN ausgeführt.

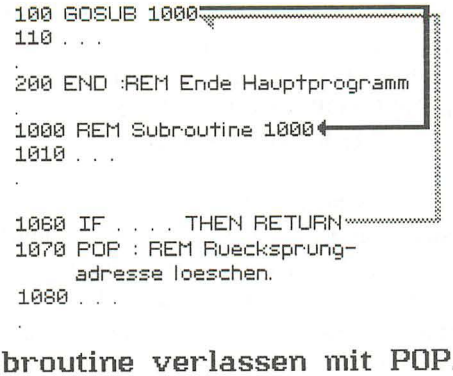
Trifft der APPLESOFT-Interpreter auf ein RETURN, ohne vorher ein GOSUB durchgeführt zu haben, so weiß er nicht wohin er zurückspringen soll, da er keine Rücksprungadresse hat. Er gibt deshalb die Fehlermeldung „RETURN WITHOUT GOSUB ERROR“ aus und stoppt das Programm.

Umgekehrt kann auch (z.B. mit GOTO) aus einem Unterprogramm herausgesprungen werden. In diesem Fall wird das zugehörige RETURN nicht ausgeführt, und die Rücksprungadresse bleibt vorhanden. Damit bei geschachtelten GOSUBs nicht anschließend der falsche Rücksprung an die nicht mehr aktuelle Adresse ausgeführt wird, kann man diese nicht mehr benötigte Rücksprungadresse durch den Befehl POP löschen:

```
10 GOSUB 100
20 PRINT  : PRINT  "Hauptprogramm"
30 PRINT  "ENDE"
40 END
100 REM - SUB100
110 POP:GOTO 20
120 PRINT  "SUB100"
130 RETURN
RUN
```

Hauptprogramm
ENDE

Die Befehle POP : GOTO 20 löschen die Rücksprungadresse (10) und springen ins Hauptprogramm. Solche Sprünge führen aber wieder zur unübersichtlichen Spagettistruktur, und sollten deshalb vermieden werden.



Subroutine verlassen mit POP.

3.7.3 Programmschleifen, FOR ... NEXT

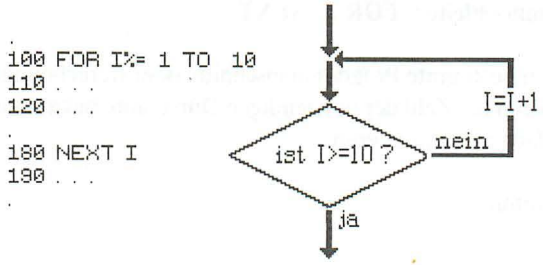
Häufig werden bestimmte Programmabschnitte oder Berechnungen wiederholt gebraucht. Wenn die Zahl der notwendigen Durchläufe bekannt ist, so läßt sich dies sehr einfach programmieren.

Beispielprogramm:

```
5  PRINT
10 FOR I = 0 TO 10
 20  A = I ↑ 2
30 PRINT  A
40 NEXT I
RUN
```

0
1
4
9
16
25
36
48
64
81
100

Das Programm berechnet die Quadratzahlen von 0 bis 10 und gibt diese aus. Die Schleife wird erzeugt durch die Kombination FOR ... TO ... NEXT.
Der erste Teil FOR ... TO bezeichnet die Schleifengrenzen und die Zählvariable, in diesem Fall I. FOR I = 0 TO 10 bedeutet:
Mache eine Scheife, zähle die Wiederholungen von Null an, höre auf, wenn der Zähler größer als 10 ist.
Zum ersten Teil der Schleife gehört der zweite Teil, das NEXT. NEXT bezeichnet die Stelle, an der die Schleife fertig ist, ähnlich wie RETURN bei GOSUB, und setzt die Zählvariable (hier I) auf ihren neuen Wert (hier I+1). Im Programm wurde nach dem NEXT in Zeile 40 der Name der Zählvariablen I erwähnt. Das ist nicht notwendig, ein NEXT ohne Zusatz springt immer zum letzten FOR .. TO im Programmablauf zurück. Für den Programmierer ist es aber bei mehreren Schleifen einfacher zu sehen, welches NEXT zu welchem FOR .. TO gehört, wenn beim NEXT die dazugehörige Zählvariable angegeben ist.



Einfache FOR . . . NEXT Scheife.

Bei FOR .. TO wurde keine Schrittbreite angegeben. In diesem Fall wird die Zählvariable (hier I) nach jedem Durchgang um eins erhöht. Mit Hilfe des Wortes STEP (= Schritt) kann die Schrittbreite verändert werden:

```
5 PRINT
10 FOR I = 0 TO 10 STEP 2
20  A = I ↑ 2
30 PRINT  A
40 NEXT I
RUN
```

0
4
16
36
64
100

Hier wird die Zählvariable nach jedem Durchgang um 2 erhöht. Deshalb werden nur die Quadrate der Zahlen 0, 2, 4, 6, 8, 10 berechnet. Der Wert der Schrittbreite STEP kann auch negative Werte annehmen. In diesem Fall wird rückwärts gezählt. Deshalb muß bei sinnvollem Gebrauch der Anfangszählwert höher sein als der Endwert:

```
10 FOR I = 1 TO 5 STEP 0
20 PRINT  "HALLO"
30 NEXT
```

Das Programm wird unendlich oft ausgeführt, da der Endwert 5, der als Kriterium für den Schleifenabbruch dient, nie erreicht werden kann, wenn ständig zum Anfangswert 1 nur die Zahl 0 addiert wird. Sollten Sie dieses Programm trotzdem laufen lassen, so können Sie es mit CONTROL-C gefolgt von RETURN unterbrechen.

Eine FOR ... NEXT Schleife wird mindestens einmal durchlaufen, da die Prüfung, ob die Zählvariable außerhalb des angegebenen Zählbereichs (von .. bis ..) liegt, erst bei NEXT stattfindet.
Die Tabelle zeigt, wie oft eine Schleife durchlaufen wird, abhängig von den angegebenen Parametern:

FOR Untere Grenze	TO Obere Grenze	STEP Schrittbreite	Durchläufe
FOR I = 0	TO 5	STEP 1	6
1	5	1	5
1	5	2	3
1	5	5	1
0	0	0	unendlich
1	5	0	unendlich
0	0	1	1
0	5	-1	1
0	-5	-1	6
5	1	-1	5
5	1	1	1

Die Schleife wird erst dann nicht mehr durchlaufen, wenn die Zählvariable außerhalb des in FOR ... TO ... angegebenen Zahlenbereichs liegt.

Sowohl die Grenzwerte für die FOR ... NEXT Schleife als auch die Zählvariable können durch Ausdrücke berechnet werden. Diese Darstellung ist allerdings schwer lesbar. Die Werte sollten besser in vorangestellten Zeilen berechnet werden.

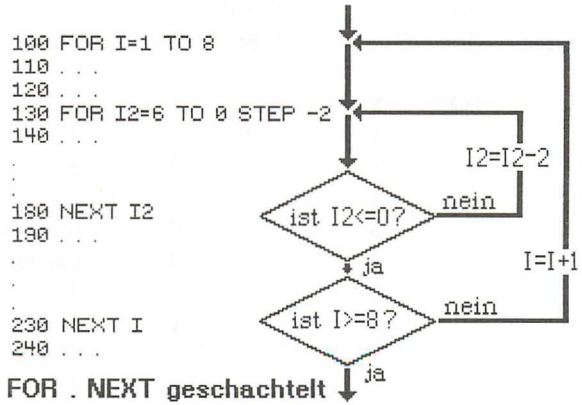
schlecht:

```
100 FOR A= ABS(A(I)/2) TO ABS(A(I+1)/2) STEP B*(A(I)-A(I+1))
. . .
```

besser:

```
100 UG=ABS(A(I)/2): REM Untergrenze
110 OG=ABS(A(I+1)/2): REM Obergrenze
120 SB=B*(A(I)-A(I+1)):REM Schrittbreite
130 FOR A= UG TO OG STEP SB
. . .
```

FOR ... NEXT Schleifen können in maximal 10 Ebenen geschachtelt werden. Wie bei GOSUBs dürfen sich die einzelnen Schleifen nicht überschneiden. Wird innerhalb einer ersten Schleife eine zweite Schleife begonnen, so muß diese beendet sein, bevor die erste beendet werden kann.
Auch bei geschachtelten Schleifen kann bei NEXT die Zählvariable weggelassen werden. Durch die obige Bedingung ist die Zuordnung von NEXT zum



FOR . NEXT geschachtelt

dazugehörigen FOR ... TO eindeutig. Für die Lesbarkeit des Programms ist die Angabe der Zählvariablen nach dem NEXT allerdings sehr nützlich.

Geschachtelte Schleifen können auf einer Zeile beendet werden; wenn die Zählvariablen hinter den NEXTs angegeben werden, müssen diese in der richtigen Reihenfolge stehen:

```
10 FOR A = 1 TO 5
20 FOR B = 3 TO 9
30 FOR C = 5 TO 2 STEP -1
40 PRINT  A,B,C
50 NEXT : NEXT : NEXT
```

Drei FOR ... NEXT Schleifen sind hier ineinander geschachtelt.

Für Zeile 50 sind noch folgende Schreibweisen möglich:

```
50  NEXT C: NEXT B: NEXT A        oder        50  NEXT C,B,A
```

3.7.4 Bedingte Verzweigungen, IF ... THEN

Die IF ... THEN Befehlskombination ist wie beispielsweise GOTO selbsterklärend. Die Übersetzung von IF ... THEN bedeutet WENN ... DANN .
Hinter IF steht ein Vergleich (z.B. A < 20); trifft dieser Vergleich zu, dann wird der oder werden die Befehle ausgeführt, die hinter THEN noch in der gleichen Zeile stehen.

```
10 A = 12 : PRINT
20 IF A < 20 THEN PRINT  "A ist kleiner als 20"
30 PRINT  "Ende des Vergleichs"
RUN
```

A ist kleiner als 20
Ende des Vergleichs

```
10 A = 22 : PRINT
20 IF A < 20 THEN PRINT  "A ist kleiner als 20"
30 PRINT  "Ende des Vergleichs"
RUN
```

Ende des Vergleichs

Der Befehl hinter THEN wird nur im ersten Fall (A = 12) durchgeführt. Im zweiten Fall fährt das Programm mit Zeile 30 fort. Ist hinter dem Wort THEN nicht genügend Platz für die Befehle, die im Falle des Zutreffens des Vergleichs ausgeführt werden sollen (die Zeile ist schließlich auf 239 Zeichen begrenzt), so muß man sich mit einer GOTO- oder GOSUB-Umschreibung behelfen.
Das obige Programm kann beispielsweise umgeschrieben werden zu:

```
10 A = 12 : PRINT
20 IF A < 20 THEN GOSUB 100
30 PRINT  "Ende des Vergleichs"
40 END
100 REM - SUB100, Ergebnisdarstellung
110 PRINT  "A ist kleiner als 20"
120  RETURN
RUN
```

A ist kleiner als 20
Ende des Vergleichs

Diese Programmgestaltung empfiehlt sich, wenn viele Befehle nach dem THEN auszuführen sind.
Die IF ... THEN Kombinationen können beliebig tief geschachtelt werden. Im Gegensatz zu den Schachtelungen von GOSUBs oder FOR ... NEXT Schleifen gibt es keine weiteren Bedingungen dafür. Einzig die vom Benutzer gewünschte Entscheidungslogik ist für die Gestaltung maßgebend.

Soll anschließend an das THEN ein GOTO verwendet werden, so kann entweder das THEN oder das GOTO weggelassen werden:

```
100 IF A > 20 THEN GOTO 200      und
100 IF A > 20 THEN 200           und
100 IF A > 20 GOTO 200
```

sind in ihrer Bedeutung identisch.

3.8 Ein- und Ausgabeprogramme

Befehle:

Eingabe: **INPUT, GET, VAL( ),**

Ausgabe: **PRINT, SPC( ), HOME, TAB( ), VTAB, HTAB, INVERSE, NORMAL, FLASH**
Allg.: **SPEED=, CHR$( ), ASC( )**

Die Ein- und Ausgabe in bzw. aus dem Computer kann über mehrere Kanäle erfolgen. Die wichtigsten sind die Tastatur, der Monitor und der Diskettenspeicher. Aber auch über die seriellen Anschlüsse und den Anschluß für Joysticks oder Maus können Daten irgendwelcher Art ein- oder ausgegeben werden.

Wird der Computer eingeschaltet, so interessiert er sich zunächst für das Diskettenlaufwerk, im Anschluß daran für die Tastatur als Eingabekanal, den Bildschirm als Ausgabemedium (falls er von der Bootdiskette keine anderen Anweisungen bekommt). In diesem Abschnitt werden nur Bildschirmausgabe und Tastatureingabe besprochen. Die Ein- und Ausgabe über andere Kanäle geschieht in APPLESOFT ganz ähnlich. Entsprechende Erklärungen werden in den nachfolgenden Abschnitten gegeben.
Die dargestellten Programme sind für die Verwendung der 80-Zeichen-Darstellung ausgelegt. Bei 40-Zeichen-Darstellung kann sich das Ausgabebild ändern. Wird als Ausgabegerät ein Drucker verwendet, so ist es möglich, daß die Darstellung auf Bildschirm und Drucker unterschiedlich ausfällt, jenachdem ob der verwendete Drucker gewisse Zeichen, z.B. TAB( ), anders interpretiert. Vor der Eingabe eines Programmes sollte das möglicherweise vorhandene alte Programm mit dem Befehl NEW (= NEU) gelöscht werden.

3.8.1 Die Eingabebefehle

3.8.1.1 Der INPUT Befehl

In APPLESOFT-BASIC bestehen zwei Eingabebefehle : INPUT und GET. Der wichtigere von beiden ist INPUT. Die einfachste Form ist

```
10 INPUT X$
```

INPUT kann nicht als Einzelanweisung verwendet werden, sondern muß Teil eines Programmes sein. Die dargestellte Anweisung erzeugt auf dem Bildschirm ein Fragezeichen und erwartet eine Eingabe über die Tastatur. Die Eingabe wird mit dem Drücken der RETURN-Taste abgeschlossen und darf maximal

255 Zeichen lang sein. Nach der Eingabe enthält die angegebene Variable (hier X$) die eingegebenen Informationen.
Statt eines Textes können auch Zahlen eingegeben werden, zum Beispiel:

```
10 INPUT A
20 INPUT B%
```

Diese beiden INPUTs haben aber einen wichtigen Nachteil: Der APPLESOFT Interpreter akzeptiert keine Buchstaben oder andere „Nichtzahlen" als Eingabe und anwortet mit der Fehlermeldung '? REENTER". Da man aber nie ausschließen kann, daß der Benutzer des Programms doch einmal einen Buchstaben statt einer Zahl eingibt und so das Programm zum Absturz bringt, kann man eine solche Eingabe umschreiben durch:

```
10 INPUT A$: A = VAL( A$ )
```

Der neue Befehl VAL( ) berechnet den Wert der Textvariablen in der Klammer. VAL ist die Abkürzung von Value: der Wert. Auf diese Weise kann man die Fehlermeldung vermeiden, denn INPUT A$ akzeptiert auch Zahlen. Das erste Zeichen des Strings muß eine Zahl sein, andernfalls errechnet VAL( ) den Wert 0. Jedes folgende Zeichen wird getestet, bis die erste Nichtzahl gefunden wird. Dieses Zeichen und alle folgenden werden ignoriert, der erste gültige Teil des Strings in eine Festkomma- oder Intergerzahl umgewandelt.

Der Befehl INPUT erlaubt zur Eingabe eine Frage zu stellen:

```
10 INPUT "Bitte den Namen eingeben :" ; A$
RUN
```

Bitte den Namen eingeben:

Dazu wird hinter INPUT ein Text in Anführungszeichen angegeben, anschließend ein Strichpunkt und der Name der Inputvariablen. Der Doppelpunkt in Zeile 10 trennt nicht zwei Befehle, sondern er wird als zum Text gehörend interpretiert, weil er zwischen Anführungszeichen steht.
Bei dieser Form des INPUT's wird kein Fragezeichen mehr dargestellt. Dies erlaubt auch den INPUT ohne Fragezeichen und Frage:

```
10 INPUT ""; A$
```

Innerhalb einer INPUT-Anweisung können auch mehrere Eingaben gemacht werden:

```
INPUT "Vorname <RETURN> Nachname";V$,N$
```

Der Hinweis <RETURN> soll den Benutzer daran erinnern, Vor- und Nachnamen durch das Drücken der RETURN-Taste zu trennen und nicht durch ein Leerzeichen. Außer bei Eingaben großer Zahlenmengen ist es aber in den meisten Fällen besser, für jede Eingabe einen eigenen INPUT vorzusehen.

3.8.1.2 Der GET Befehl

Der Eingabebefehl GET ist im Prinzip einfacher als INPUT, man sollte jedoch etwas vorsichtig mit ihm umgehen:

```
20 GET X
```

GET kann wie INPUT nur als Teil eines Programmes verwendet werden, nicht als Einzelanweisung. GET X$ wartet nur auf die Eingabe eines Zeichens, egal welches, stellt es aber nicht auf dem Bildschirm dar. Es spricht auch schon an, wenn versehentlich eine Taste gedrückt wird. Analog zu INPUT erwartet der Befehl GET X% oder GET X nur eine Zahl, bei allen anderen Eingaben erscheint die Fehlermeldung '? SYNTAX ERROR".

Steht der GET Befehl an einer wichtigen Stelle im Programm, z.B. wo eine Fehleingabe leicht zu Katastrophen führt, sollte man den eingebenen Wert prüfen:

```
10 PRINT "Zum Weitermachen bitte 'Q' drücken"
20 GET X$ : PRINT X$
30 IF X$ <> "Q" THEN 10
40 PRINT "Danke, Fortsetzung folgt"
RUN
```

Zum Weitermachen bitte 'Q' drücken"
b
Zum Weitermachen bitte 'Q' drücken"
q
Zum Weitermachen bitte 'Q' drücken"
Q
Danke, Fortsetzung folgt

In diesem Programm wird der eingegebene Buchstabe durch PRINT ausgegeben. In vielen Fällen ist es besser, auf die Wiedergabe der gedrückten Taste zu verzichten; z.B. wird der Bildschirminhalt bei falscher Eingabe nicht verändert.

3.8.2 Ausgabebefehle

3.8.2.1 Der PRINT Befehl

Der wichtigste Ausgabe Befehl ist PRINT. PRINT heißt „DRUCKEN" und stammt aus einer Zeit, wo Drucker die Standardausgabegeräte waren. Wir haben ihn in den verschiedenen Beispielprogrammen schon kennengelernt in den Formen:

PRINT X$

PRINT "HALLO"

PRINT A/2 + (45.6 * X ↑ 3)

PRINT kann in APPLESOFT abgekürzt werden durch das Fragezeichen:

PRINT "Neue Zeile" ist identisch mit ? "Neue Zeile"

Werden bei der Eingabe von Programmzeilen (mit Fragezeichen) abgekürzte PRINT Befehle verwendet, so werden sie im Listing ausgeschrieben.
Mit PRINT können mehrere Texte oder Variablen gleichzeitig ausgegeben werden. Dabei gibt es 2 Möglichkeiten:

```
10 A$ = "BIER" : B$ = "DOSE"
20 PRINT
30 PRINT A$,B$
40 PRINT
50 PRINT A$;B$
RUN
```

BIER DOSE

BIERDOSE

Im ersten Fall sind A$ und B$ durch ein Komma getrennt, im zweiten Fall durch einen Strichpunkt. Die Trennung durch Komma (oder Kommas bei mehr als 2

Variablen) bewirkt, daß ein Ausdruck immer 16 Zeichen rechts vom Anfang des vorhergehenden beginnt, außer der vorhergehende Ausdruck war mehr als 15 Zeichen lang. Dann erfolgt der zweite nochmals um 16 Stellen weiter rechts. Die Darstellungen mit PRINT, getrennt durch Kommas, erfolgen auf den Positionen 1, 17, 33, 49 und 65.

Bei der Trennung der Variablen oder Texte beim Befehl PRINT durch einen Strichpunkt werden diese direkt nacheinander ausgegeben.
Beim Überschreiten der 80 Zeichen pro Zeile wird auf der nächsten Zeile mit der Ausgabe fortgefahren.

Die erste Art der Variablentrennung mit Kommas eignet sich vor allem für die Darstellung von Zahlenreihen:

```
 5 PRINT
10 FOR I = 1 TO 6
20 FOR J = 1 TO 4
30 PRINT  I + J / 4 ,
40 NEXT J
50 PRINT
60 NEXT I
RUN
```

1.25	1.5	1.75	2
2.25	2.5	2.75	3
3.25	3.5	3.75	4
4.25	4.5	4.75	5
5.25	5.5	5.75	6
6.25	6.5	6.75	7

Wie in dem Beispiel gezeigt, funktioniert die Tabellierung durch das Komma auch bei getrennten PRINT Befehlen. Das Komma nach dem zu druckenden Wert (hier I + J/4) bewirkt beim nächsten PRINT Befehl die Ausgabe 16 Zeichen nach dem Beginn des Vorhergehenden.
Der Abschluß eines PRINT Befehls mit Semikolon bewirkt, daß der nächste PRINT Befehl direkt im Anschluß an die ausgegeben Zeichen druckt.

```
PRINT "HALLI "; : PRINT "HALLO"
```

HALLI HALLO

Das Freizeichen zwischen HALLI und HALLO ist in dem HALLI-String enthalten.

In den folgenden Zeilen ist die neue Schreibposition, d.h. der Ort, an dem der neue Text (beim nächsten PRINT) ausgegeben wird, mit dem Zeichen _ gekennzeichnet.

```
PRINT "TEXT"          PRINT "TEXT";          PRINT "TEXT",
TEXT                  TEXT_                  TEXT            _
_
```

Natürlich können mit PRINT Variable und Ausdrücke verschiedener Typen gemischt werden. Das wird vor allem bei der Darstellung von Ergebnissen angewendet:

```
5  PRINT
10 INPUT "Erster  Summand : " ; X
20 INPUT "Zweiter Summand : " ; Y
30 PRINT
40 PRINT "Die Summe von ";X;" und ";Y;" ist ";X+Y
RUN
```

Erster Summand: 12345
Zweiter Summand: 7655

Die Summe von 12345 und 7655 ist 20000

3.8.2.2 Der Befehl SPC( )

Will man einen Ausdruck nicht am Zeilenanfang dargestellt haben, sondern z.B. eingerückt, so ist das möglich mit:

```
5  PRINT : Z$ = "Zeilenanfang"
10 PRINT Z$
20 PRINT "       "; Z$ ; ", eingerückt."
RUN
```

Zeilenanfang
    Zeilenanfang, eingerückt.

Einfacher ist die Formulierung mit dem Befehl SPC(...). SPC ist die Abkürzung von Space (der Raum) und bewirkt, daß die in Klammern hinter SPC(...) angegebene Zahl von Leerzeichen ausgegeben wird:

```
5  PRINT
10 FOR I = 0 TO 4
20 PRINT SPC (5 * I) ; 5 * I ; " Leerzeichen"
30 NEXT I
RUN
```

```
0 Leerzeichen
     5 Leerzeichen
          10 Leerzeichen
               15 Leerzeichen
                    20 Leerzeichen
```

Hier wird die Anzahl der Leerzeichen in der Klammer hinter SPC berechnet (5 * I); anschließend wird der dazugehörige Text ausgegeben.

SPC zählt immer von der aktuellen Cursorposition aus, nicht vom Zeilenanfang; wurde beispielsweise vorher ein PRINT mit Strichpunkt ausgeführt, so wird vom ersten Freizeichen nach dem ausgegebenen Text aus gezählt:

PRINT "1.Text"; : PRINT SPC (6);"Text 2"

```
1. Text          Text 2
```

6 Freizeichen zwischen '1.Text' und 'Text 2'
SPC( ) kann Argumente zwischen 0 und 255 besitzen.

3.8.2.3 Die Befehle TAB, HTAB UND VTAB

Der Befehl TAB (Tabulator) funktioniert ähnlich wie auf der Schreibmaschine. Analog zu SPC wird in der Klammer hinter TAB die Anzahl der Stellen angegeben, die freigehalten werden sollen. Im Gegensatz zu SPC zählt TAB jedoch vom Zeilenanfang aus. Die letzte Darstellung kann also statt mit SPC auch mit TAB geschrieben werden:

PRINT "1.Text"; TAB(12);"Text 2"

```
1.Text          Text 2
```

„Text 2" steht in Spalte 13 bis 18.

Der Befehl TAB(12); setzt den Beginn des zweiten Texts auf die Position 13, vom Zeilenanfang aus gesehen. Die Stellenzahl oder der Abstand vom Zeilenanfang bei TAB kann ebenso wie bei SPC berechnet werden.

Für die Darstellung von Tabellen kommen also 3 Befehle in Betracht: TAB( );, SPC( ); , und PRINT mit Komma.

Die einfachste Programierung geschieht mit PRINT ... , die aber dann zu ungünstiger Darstellung führt, wenn die dargestellten Zeichenketten mehr als 15 Zeichen lang ist.

```
10 ? : A= 122: ? A, A, A, A
RUN

122             122             122             122
```

Der Befehl SPC( ); kann gut verwendet werden, wenn die Zeichenketten immmer gleich lang sind. Sonst muß das zu SPC( ); in Klammern gehörige Argument (die Zahl der Freizeichen) in Abhängigkeit der Länge der Zeichenketten errechnet werden:

```
10 PRINT : A = 122
20 FOR I=1 TO 4 :PRINT A; SPC(13); : NEXT
RUN

122             122             122             122
```

Die Darstellung von Tabellen mit TAB( ); ist dann günstig, wenn die Zeichenketten unterschiedlich lang sind:

```
 5 PRINT
10 FOR I = 1 TO 5
20 FOR J = 1 TO 4
30 PRINT10 / I / J; TAB (12 * J );
40 NEXT J
50 PRINT
60 NEXT I
RUN

10              5               3.33333333      2.5
5               2.5             1.66666667      1.25
3.33333333      1.66666667      1.11111111      .833333333
2.5             1.25            .833333333      .625
2               1               .666666667      .5
```

Allerdings werden zu lange Zeichenketten teilweise überschrieben, wenn sie bis zur vorgesehenen Stelle des nächsten Textes reichen:

```
PRINT "DIES IST EIN LANGER TEXT" ; TAB(22) ; "2. TEXT"

DIES IST EIN LANGER TE2.TEXT
```

TAB( ) kann wie SPC( ) Argumente zwischen 0 und 255 besitzen. TAB(0) bewirkt einen Sprung von 256 Zeichen.

Der Bildschirm besteht aus 24 Zeilen zu je 80 (oder 40) Zeichen. Mit Hilfe der Befehle VTAB (vertical Tabulator) und HTAB (horizontal Tabulator) läßt sich der Cursor unabhängig von seiner momentanen Position plazieren:

```
HTAB 20: VTAB 7
```

plaziert den Cursor in die 7. Zeile und die 20. Spalte. Für VTAB sind die Argumente 1 bis 24 einschließlich zulässig. Für HTAB die Argumente 1 bis 80 einschließlich.

3.8.2.4 Die Befehle INVERSE, FLASH, NORMAL UND SPEED=

Der Befehl INVERSE bewirkt die „negative" Darstellung auf dem Bildschirm. Alles, was nach dem Befehl INVERSE auf dem Bildschirm erscheint, wird dunkel auf hellem Untergrund dargestellt. Mit dem Befehl NORMAL wird wieder die übliche Darstellung hell auf dunklem Hintergrund bewirkt. Invertiert dargestellter Text wird dabei jedoch nicht gelöscht. Die invertierte Darstellung eignet sich vor allem für wichtige Texte oder Zahlen. Eine einzelne Zahl kann auf normalem Bildschirm erzeugt werden durch:

INVERSE: ? 450 : NORMAL

Anmerkung: Der Apple //c besitzt 2 Zeichensätze. Bestimmte (kleine) Buchstaben werden deshalb im INVERSE-Modus anders dargestellt als im NORMAL-Modus. Im folgenden Beispielprogramm wird dies deutlich:

```
 10  REM  Demoprogramm für Zeichensatz
 20  HOME
 30  SPEED= 100
100  PRINT : PRINT "Normaler Zeichensatz:": POKE 49166,0
110  GOSUB 300
120  NORMAL : PRINT
130  FOR I = 1 TO 1000: NEXT I: HOME
150  PRINT  CHR$ (27);
200  PRINT: PRINT "Alternativer Zeichensatz:": POKE 49167,0
210  GOSUB 300
```

```
220  NORMAL: PRINT  CHR$ (24);: REM  Mauszeichen ausschalten
230  SPEED= 255
240  END
300  NORMAL : PRINT : PRINT "    NORMAL": GOSUB 400
310  NORMAL : PRINT : PRINT "   INVERSE": INVERSE:GOSUB 400
320  NORMAL : PRINT : PRINT "    FLASH": FLASH : GOSUB 400
340  RETURN
400  FOR I = 32 TO 127: PRINT  CHR$ (I);: NEXT I: RETURN
```

Der normale Zeichensatz wird mit PR#0, ESC-CONTROL-Q oder POKE 49166,0 oder durch Einschalten des Computers aktiviert.

Der alternative Zeichensatz wird durch PR#3, ESC-4, ESC-8 oder POKE 49167,0 eingeschaltet. Ist der alternative Zeichensatz aktiv, so kann mit ESC-4 und ESC-8 zwischen 40 und 80 Zeichendarstellung gewechselt werden, ohne Auswirkung auf den Zeichensatz.
Bei alternativem Zeichensatz werden nach CHR$(27); anstelle der inversen Großbuchstaben die Mauszeichen dargestellt. Nach CHR$(24); werden wieder inverse Großbuchstaben anstelle der Mauszeichen aktiv.
Die Zeichensätze sind im Anhang dargestellt.

Im Programm wurden noch nicht behandelte Befehle verwendet, die jetzt hier in der Folge erklärt sind:
Die Aufmerksamkeit des Betrachters auf bestimmte Teile des Bildschirms kann mit dem Befehl FLASH gesteigert werden. Dieser Befehl läßt alle im FLASH Modus ausgegeben Zeichen in der Darstellung zwischen normal und invertiert wechseln. Genau wie INVERSE wird FLASH durch den Befehl NORMAL rückgängig gemacht.

Bei manchen Programmen ist es notwendig, die Arbeitsgeschwindigkeit des Apple //c zu reduzieren (beispielweise für manche Drucker bei der Datenausgabe). Das geschieht mit dem Befehl SPEED= :

```
SPEED= 100
```

Mit SPEED= 100 wird der Computer deutlich langsamer als normal. SPEED kann Werte zwischen 0 und 255 annehmen. Mit SPEED= 255 ist der Computer am schnellsten. Dieser Wert stellt sich automatisch beim Einschalten ein. Die Geschwindigkeit kann überprüft werden mit Listings. Listings oder Daten, die etwas langsamer über den Bildschirm laufen (z.B. SPEED= 200), sind für das Auge angenehmer.

Mit dem Apple können bestimmte Bildschirmausschnitte (Windows) festgelegt werden, in denen die Datenausgabe möglich ist. Außerhalb dieser Windows

bleiben vorhandene Informationen bestehen, und können erst durch Verändern der Windowgrenzen überschrieben werden. Da in APPLESOFT keine Befehle für diese Manipulationen existieren, müssen diese über direkte Speicherzugriffe geschehen. Die Angaben dazu stehen im Anhang.

3.8.2.5 Die Befehle CHR$ UND ASC( )

Die Tastatur des Apple //c kann manche Zeichen nicht erzeugen: Diese Zeichen können mit dem Befehl CHR$(...) generiert werden:
Wie schon erwähnt, existiert für jedes Zeichen eine Nummer. Diese Numerierung und Festlegung der möglichen Zeichen, also auch solcher, die auf einer normalen Schreibmaschinentastatur nicht zu finden sind, heißt ASCII-Code. (ASCII = American Standard Code for Information Interchange). Der Computer selbst erkennt alle Zeichen nur an den dazugehörigen ASCII-Nummern. Die US-Tastatur, zu der Sie mit dem rechten Schalter oberhalb der Tastatur von der deutschen Tastatur wechseln können, ist die ASCII-Tastatur. Wenn Sie programmieren, sollten Sie diese ASCII-Tastatur verwenden.
Zeichen, die nicht auf der Tastatur vertreten sind, werden durch CHR$(ASCII-Nummer) erzeugt. Wenn Sie beispielsweise eingeben:

```
PRINT CHR$(65)
A
```

so erscheint ein A auf dem Bildschirm. Natürlich ist es bei diesem Zeichen sinnvoller, einfach die Taste A zu drücken. Der „reziproke" Befehl zu CHR$( ) ist ASC( ) :
PRINT ASC(A) liefert den Wert 65, also die zugehörige ASCII-Nummer.

Die Zeichen von A bis Z beispielsweise haben die ASCII-Nummern 65 bis 90 (ohne Umlaute). Die Nummern der anderen Zeichen des ASCII-Codes können Sie im Anhang finden.
Wichtige nichtdruckende Zeichen sind BELL (Control-G) oder Control-D. Innerhalb von Programmen kann z.B. BELL erzeugt werden durch:

```
PRINT CHR$(7)
```

Dabei ertönt der Lautsprecher. Dieses Zeichen ist in einem Programm beispielsweise als Warnung bei Fehleingabe sinnvoll.

3.8.3 Die Gestaltung von Ein- und Ausgabeprogrammen

3.8.3.1 Eingabeprogramme

Bei Eingabeprogrammen, oft Abschnitte oder Unterprogramme anderer Programme, wird der Computer mit Daten gefüttert. Dies ist normalerweise der Ort, wo die meisten Fehler auftreten. Um die Fehlerquote zu reduzieren, sind zwei Grundregeln zu beachten:

1. Eingabeprogramme sind so zu gestalten, daß möglichst wenig Fehler gemacht werden können.
2. Eingaben müssen korrigierbar sein.

Um diese Regeln zu erfüllen gibt es folgende Möglichkeiten:
a) Klare Fragen stellen; der Benutzer muß möglichst gut informiert werden, welche Eingabe von ihm erwartet wird.
b) Müssen mehrere Eingaben erfolgen, so sollten diese direkt nacheinander abgefragt und erst anschließend weiterverarbeitet werden. (Beispiel: Vorname, Nachname, Straße, Ort, Telefonnr.)
c) Plausibilitätstests so früh wie möglich durchführen, und gegebenfalls den Benutzer am besten akustisch warnen (z.B. Prüfen, ob die Telefonnummer nur aus Zahlen besteht oder Buchstaben enthält).
d) Die eingegebenen Daten auf dem Bildschirm längere Zeit stehen lassen, nicht sofort löschen, vor allem bei zusammengehörenden Eingaben.
e) Am Ende einer Eingabegruppe die bisherigen Eingaben nochmals darstellen und Möglichkeiten zur Korrektur geben.

Beispiel für ein Standardeingabeprogramm:

```
100  REM     - Eingabeprogramm ; E = Anzahl Fragen
110  REM     - Antworten in E$(E) ; Fragen in F$(E)
120  REM  Titel in T1$,T2$,T3$
130  HOME : IF  LEN (T1$) > 0 THEN  PRINT T1$
140  IF  LEN (T2$) > 0 THEN  PRINT T2$
150  IF  LEN (T3$) > 0 THEN  PRINT T3$
160  PRINT : PRINT :V% =  PEEK (37)
170  FOR I9 = 1 TO E
180    PRINT  SPC( 30 -  LEN (F$(I9)));F$(I9);":";: INPUT
       "";E$(I9)
190  NEXT I9
200  PRINT : INPUT "Sind alle Eingaben richtig?, N=nein, sonst
     ja :";X$
210  IF X$ = "N" OR X$ = "n" THEN 230
```

```
220  T1$ = "":T2$ = "":T3$ = "": FOR I9 = 1 TO E:F$(I9) = "":
      NEXT I9:I9 = 0: HOME : RETURN
230  VTAB  PEEK (37): PRINT  SPC( 60): VTAB V% + 1: REM  -
      Korrektur
240  FOR I9 = 1 TO E
250    HTAB 32: INPUT "";X$: IF  LEN (X$) <  > 0 THEN E$(I9)
        = X$
260    VTAB  PEEK (37): HTAB 32: PRINT E$(I9)
270  NEXT I9: GOTO 200
```

Das ist ein Beispiel für ein Eingabeprogramm. Es kann separat auf Diskette gespeichert werden und z.B. mit dem DOS 3.3 Programm RENUMBER in ein eigenes Programm eingesetzt und dort mit GOSUB aufgerufen werden. Das Eingabeprogramm verwendet folgende Übergabeparameter:

T1$,T2$,T3$:	Titel
E :	Anzahl Fragen, die beantwortet werden sollen. (Maximal 9)
F$(1) bis F$(E):	Texte zu den Fragen 1 bis E mit maximal 30 Zeichen pro Frage
E$(1) bis E$(E):	Antworten zu den Fragen 1 bis E.

Das Programm löscht den Bildschirm und schreibt die Titel mit zwei Freilinien (Zeile 100 bis 160).
Anschließend werden die Fragen gestellt und die Anworten entgegengenommen. Danach erfolgt in Zeile 200 der Hinweis auf die Korrekturmöglichkeit.
Die Korrektur läßt die vorher gestellten Fragen mit ihren bereits gegebenen Anworten auf dem Bildschirm, und setzt den Cursor auf den ersten Buchstaben der jeweiligen Antwort. Mit RETURN kann diese akzeptiert werden, andernfalls wird sie überschrieben.
Sind alle Fragen richtig beantwortet, so werden die verwendeten Fragen und weitere Parameter gelöscht und der RETURN zum aufrufenden Programmteil durchgeführt. Dort können die Anworten weiterverarbeitet werden, die in E$(1) bis E$(E) vorhanden sind.

3.8.3.2 Ausgabeprogramme

Ausgabeprogramme sind in der Regel einfacher zu schreiben als Eingabeprogramme, vor allem deshalb, weil der Dialog zwischen Computer und Benutzer wegfällt.

In diesem Abschnitt wird die Ausgabe auf den Bildschirm beschrieben; soll der Drucker als Ausgabekanal verwendet werden, so gelten andere Regeln, weil der Drucker oft mehr Zeichen auf einer Zeile darstellen kann als der Bildschirm, weil einmal ausgedruckte Zeichen nicht mehr verschwinden, und weil die Druckerausgabe sich an einem anderen Format orientiert (in der Regel DIN A4), der Bildschirm aber maximal 24 x 80 Zeichen umfaßt.

Bei der Datenausgabe auf den Bildschirm gilt es folgende Richtlinien zu beachten:
a) Die Daten nicht „auf einem Haufen" darstellen.
b) Spezifikation der Daten angeben, (z.B. Konto-nr.: , Tel-nr.: etc).
c) Viele Zahlen am besten in Tabellenform darstellen.
d) Vermeiden, daß die Daten wieder aus dem Bildschirm wandern, z.B. nach der Darstellung von 22 Zeilen ein GET X$ einfügen, danach HOME, und wieder 22 Zeilen darstellen.

Ein Beispiel für ein Ausgabeprogramm ist in Abschnitt 3.16.2.2 vorhanden.

3.9 Daten im Programmtext

Befehle: **DATA, READ, RESTORE**

In diesem Abschnitt werden vor allem die Befehle DATA und READ behandelt. Diese Befehle dienen zur einfachen Speicherung von Daten innerhalb des Programmtextes.

Beispiel:

Ein Programm zur Berechnung des Ölverbrauchs eines Hauses soll den jeweiligen Monatsverbrauch mit dem Vorjahresmonatsverbrauch vergleichen und Abweichungen angeben.
Dazu müssen im Programm die 12 Vormonatsverbräuche gespeichert sein. Das kann so geschehen:

```
10 DIM V( 12 )
20 V( 1 ) =  600 : V( 2 ) =  490 : V( 3 ) =  340 : V( 4 ) =  180 :
V( 5 ) =  90 : V( 6 ) =  50 : V( 7 ) =  35 : V( 8 ) =  35 :
V( 9 ) =  80 : V( 10 ) =  170 : V( 11 ) =  360 : V( 12 ) =  520
```

Diese Methode ist aber etwas aufwendig, vor allem, wenn mehr als diese 12 Zahlen gespeichert werden müssen. Statt jeden Monatsverbrauch durch eine Variable V (Monat) auszudrücken, werden die Zahlen mit DATA gespeichert:

```
10 DATA 600, 490, 340, 180, 90, 50, 35, 35, 80, 170, 360, 520
```

Die Anweisung DATA enthält eine Reihe von Zeichengruppen, hier Integer-Zahlen, die durch Komma getrennt werden. Die Anzahl dieser Zeichengruppen ist nur durch die maximale Zeilenlänge (239 Zeichen) begrenzt.Bei Bedarf können mehrere DATA Anweisungen in mehreren Zeilen verwendet werden. Diese Darstellung ist bei gleichem Informationsgehalt wesentlich kürzer und braucht weniger Speicherplatz als in der ersten Formulierung.
Die in DATA gespeicherten Zeichen können mit dem Wort READ aufgerufen werden:

```
20 FOR I = 1 TO 12 : READ A : PRINT A;" "; : NEXT A
RUN
```

600 490 340 180 90 50 35 35 80 170 360 520

Statt Zahlen können auch Texte (in Anführungszeichen) gespeichert werden. Bei der entsprechenden READ Anweisung muß eine Stringsvariable als Argument verwendet werden, um einen „? SYNTAX ERROR“ zu vermeiden. Zahlen und Texte können auch gemischt verwendet werden.

Beispiel:

```
10 DATA "String1", 911, 928, "String2"
20 READ A$: READ A: READ B: READ B$
30 PRINT A$, A, B, B$
RUN
```

String1 911 928 String2

Der APPLESOFT-Interpreter benutzt eine interne Variable (die im Programm nicht auftaucht), um die Zahl der bereits durchgeführten READs zu speichern. Beim Starten des Programms zeigt diese Variable auf die erste Zeichengruppe nach der ersten DATA Anweisung. Nach jedem READ wird sie um eins erhöht. Wenn mehr READs durchgeführt werden, als Zeichengruppen in DATA Anweisungen vorhanden sind, erscheint die Fehlermeldung „OUT OF DATA ERROR“.

Um die in DATA Anweisungen vorhandenen Zeichengruppen mehrmals gebrauchen zu können, kann die interne Variable, die die durchgeführten READs zählt, mit dem Befehl RESTORE zurückgesetzt werden. Das erste READ nach einem RESTORE liest deshalb wieder die erste Zeichengruppe nach der ersten DATA Anweisung.

Beispiel:

```
10 DATA 2, 3, 4
20 DATA 7, 8, 9
30 FOR I= 1 TO 4: READ A: PRINT A,: NEXT A: PRINT
40 RESTORE
50 FOR I= 1 TO 4: READ A: PRINT A,: NEXT A: PRINT
RUN


2           3           4           7
2           3           4           7
```

Die Bezeichnung der mit READ gelesenen Zahl (hier A) hat keinen Einfluß, wohl aber der Variablentyp (Numerische Variable oder Textvariable).
Maßgebend für den Zugriff auf eine bestimmte DATA-Zahl oder -Zeichengruppe ist die Anzahl der bereits durchgeführten READ Befehle, unabhängig von den dabei verwendeten Variablenbezeichnungen.

Das vollständige Programm zum Vergleich des Oelverbrauchs von oben sieht beispielsweise so aus:

```
10 DATA 600, 490, 340, 180, 90, 50, 35, 35, 80, 170, 360, 520
20 HOME : PRINT "Bitte den Monat angeben, 1 bis 12"
30 INPUT "Jan=1, Feb=2, etc., Dez=12      : "; X$
40 X= VAL(X$) : IF X < 1 OR X > 12 THEN 20
50 INPUT "Monatsverbrauch in Ltr. angeben : ";V$
60 V = VAL (V$)
70 FOR I=1 TO X : READ A : NEXT I
80 PRINT : PRINT "Die Differenz zum Vorjahresmonat ist "; V - A ;" Lit
er ."
90 PRINT "Die  prozentuale Abweichung betraegt ";
   INT ((V - A) / V * 100) ; " % ."
100 PRINT : PRINT "X = Ende, RETURN = neue Rechnung" : PRINT "Bi
tte waehlen"
110 GET X$ : IF X$ <> "X" THEN RESTORE : GOTO 20
RUN
```

Bitte den Monat angeben, 1 bis 12
Jan=1, Feb=2, etc., Dez=12 : 9
Monatsverbrauch in Ltr. angeben : 100

Die Differenz zum Vorjahresmonat ist 20 Liter .
Die prozentuale Abweichung betraegt 25 % .

X = Ende, RETURN = neue Rechnung
Bitte waehlen

Der zu berechnende Monat wird durch eine Zahl zwischen 1 und 12 angegeben (1 für Januar, 12 für Dezember). Andere Eingaben werden in Zeile 40 gefiltert, gegebenenfalls wird neu angefangen. Zu dem Monat wird der Verbrauch angegeben.
Das Programm ermittelt den Vorjahresverbrauch durch wiederholtes Lesen der DATA-Zahlen mit dem Befehl READ in Zeile 70. Im Beispiel wird der READ-Befehl 9 mal durchgeführt und dabei die ersten 9 Zahlen hinter DATA gelesen. Bis zur neunten Zahl wird jeder gelesene Wert A durch den nächsten überschrieben. Nach Zeile 70 enthält die Variable A den neunten Wert (Vorjahresverbrauch für September). In Zeile 80 und 90 werden die Resultate berechnet und ausgegeben.
In der letzten Zeile kann das Programm durch Eingabe von „X" beendet werden. Andernfalls wird mit dem Befehl RESTORE die interne Variable, die die bereits durchgeführten READ-Befehle zählt (hier 9), auf 0 zurückgesetzt. Diese Variable bestimmt, welche der DATA Zahlen beim nächsten READ-Befehl gelesen wird.

Bemerkung zur Programmgestaltung:
Das Programm ist nicht strukturiert. Das ganze Programm ist, wie viele andere Programme, im Vergleich zu den eigentlichen Rechnungsausdrücken in den Zeilen 80 und 90 sehr lang.
Bei komplizierteren Rechnungsausdrücken ist es sinnvoll, die Berechnung von der Ergebnisdarstellung zu trennen.

3.10 Funktionen

Die APPLESOFT-Funktionen sind in ihrer Darstellung und Arbeitweise sehr an die mathematischen Funktionen angelehnt. Zusätzlich existieren einige Text-Funktionen, die ähnlich den numerischen aufgebaut sind, sich aber auf Texte oder Strings beschränken.

3.10.1 Numerische Funktionen

Numerische Funktionen können behandelt werden wie Variable. Allerdings kann Ihnen kein Wert zugewiesen werden, sondern es werden mit Ihnen lediglich numerische Größen errechnet. Die Funktionen lassen sich aber schachteln, sofort weiterverarbeiten, direkt ausgeben oder können Variablen zugeordnet werden.
Die in APPLESOFT direkt implementierten Funktionen sind folgende:

algebraische Funktionen:

Name	Beispiel	Beschreibung
SGN	SGN(-4.4)	SGN (Signum) ermittelt das Vorzeichen einer Zahl mögliche Werte : +1, -1
ABS	ABS(B%)	ABS (Absolut) ermittelt den Betrag einer Zahl Wertebereich: 0 bis 10 hoch 38
INT	INT(15.33)	INT wandelt eine Gleitkommazahl in eine Integerzahl. Wertebereich: -32767 bis 32767
SQR	SQR(16)	SQR berechnet die Quadratwurzel einer Zahl Wertebereich: 0 bis 10 hoch 19
EXP	EXP(3)	berechnet e-Funktion, e hoch Argument (e = 2.718) Wertebereich: 0 bis 10 hoch 38
LOG	LOG(6)	berechnet natürlichen Logarithmus (Basis e) Wertebereich: 0 bis ca. 87.5
RND	RND	(randomize) erzeugt eine Zufallszahl Wertebereich 0 bis 1

trigonometrische Funktionen

SIN	SIN(.4)	ermittelt den Sinus eines Winkels in Radian Wertebereich -1 bis 1

COS	COS(1.4)	ermittelt den Cosinus eines Winkels in Radian Wertebereich -1 bis 1
TAN	TAN(2)	ermittelt den Tangens eines Winkels in Radian Wertebereich +/- 10 hoch 38
ATN	ATN(.5)	ermittelt Arcustangens in Radian Wertbereich +/- PI/2

3.10.2 Funktionen definieren

Befehle: **DEF FN, FN**

Zusätzlich zu den vorhandenen Funktionen können neue definiert werden mit der Befehlskombination DEF FN.

```
10 DEF FN US(X) = 12 * SQR (X)
20 PRINT FN US(4)
RUN
```

24

DEF FN erwartet die Bezeichnung einer Funktion (hier US(X)) und eine Zuweisung mit dem Gleichheitszeichen zum Berechnungsausdruck (hier 12 * SQR(X)).
Die definierte Funktion (hier US(X)) wird aufgerufen durch FN "Funktionsname".
Die Bezeichnung der Funktionsvariablen X spielt keine Rolle. Die Funktion kann auch mit anderen Bezeichnungen der Funktionsvariablen aufgerufen werden. Erweitert man das obige Programm um Zeile 30, so erhält man:

```
10 DEF FN US(X) = 12 * SQR (X)
20 PRINT FN US(4)
30 Z= 25 : PRINT FN US(Z)
RUN
```

24
60

3.10.3 Textfunktionen

Textfunktionen beziehen sich nur auf Text- oder Stringvariable. Die Funktionen sind den numerischen sehr ähnlich aufgebaut:

Name	Beispiel	Beschreibung
ASC	ASC(S)	ermittelt die ASCII Zahl eines Zeichens
LEN	LEN(A$)	ermittelt die Länge eines Strings
VAL	VAL(B$)	ermittelt den numerischen Wert eines Strings
CHR$	CHR$(65)	wandelt eine ASCII Zahl in ein Zeichen um
STR$	STR$(12)	wandelt eine Zahl in einen String um
LEFT$	LEFT$(A$,3)	trennt (3) Zeichen vom linken Rand eines Strings
RIGHT$	RIGHT$(B$,5)	trennt (5) Zeichen vom rechten Rand eines Strings
MID$	MID$(A$,2,4)	trennt (4) Zeichen ab einem bestimmten Zeichen (2) vom Anfang eines Strings

3.11 Anweisungen an das Betriebssystem

Befehle: **CHAIN, EXEC, FLUSH, FRE, FP, IN#, INT, MAXFILES, MON, NOMON, PREFIX, PR#, -,**

(RUN, SAVE, CATALOG, INIT, LOAD, LOCK, UNLOCK, VERIFY, RENAME, DELETE)

Dieser Abschnitt behandelt das Betriebssystem und dessen Befehle mit Ausnahme derjenigen, die sich auf den Informationszugriff im File selbst beziehen.

Bisher wurden nur Befehle besprochen, die innerhalb eines BASIC-Programms Wirkungen haben. Neben dem BASIC-Programm, oft nur Programm genannt, existieren im Speicher des Apple//c weitere Programme, zum Beispiel der APPLESOFT-Interpreter, der für die vorgesehene Ausführung des BASIC-Programms sorgt.
Schließlich gibt es noch das Betriebssystem, ein Programm, das dafür sorgt, daß der Computer überhaupt läuft. Das Betriebssystem eines Computers ist in etwa vergleichbar mit dem Maschinisten auf einem Schiff. Der Maschinist bekommt vom Steuermann verschiedene Befehle, zum Beispiel „schneller fahren", oder „rückwärts". Der Maschinist setzt diese Befehle um in Handlungen, z.B. verstellt er die Einspritzpumpe, um die Maschinenleistung zu steigern, oder er sorgt dafür, daß der Motor rückwärts läuft.
Das Betriebsystem eines Computers hat ähnliche Aufgaben. Ähnlich wie der Maschinist meldet, daß die gewünschte Schiffsgeschwindigkeit zu groß ist, gibt das Betriebssystem Fehlermeldungen aus, falls der eingegebene Befehl nicht zulässig ist.
Diese Befehle heißen DOS- oder ProDOS-Befehle (DOS = Disc Operating System). Befehle an das Betriebssystem können direkt über die Tastatur eingegeben werden oder innerhalb von Programmen stehen. Es gibt DOS- oder ProDOS-Befehle, die gleich heißen wie APPLESOFT-Befehle, zum Beispiel READ, die aber eine andere Bedeutung haben. Aus diesem Grund muß zwischen APPLESOFT- und Betriebssystembefehlen unterschieden werden.

Über die Tastatur können die Befehle an das Betriebssystem direkt eingegeben werden, mit Ausnahme einiger spezieller Ausdrücke, die nur aus Programmen heraus möglich sind (OPEN, READ, WRITE etc.).
Um einen Befehl an das Betriebssystem durch ein Programm geben zu lassen, muß dieser speziell gekennzeichnet werden. Es muß ein Hinweis erfolgen, daß der nächste Befehl nicht zur Ausführung durch den Applesoft Interpreter gedacht ist, sondern vom Betriebssystem behandelt werden soll: das geschieht mit CONTROL-D. CONTROL-D ist ein nichtdruckendes Zeichen, es wird auf dem Bildschirm nicht dargestellt. Es kann auf der Tastatur hervorgerufen werden durch gleichzeitiges Drücken von CONTROL und D. Innerhalb eines Programmes wird es aber besser durch den Befehl CHR$(4) erzeugt. CHR$(4) generiert das Zeichen, dessen ASCII-Code 4 ist. Dieses Zeichen ist CONTROL-D. Ein Befehl an das Betriebssystem durch ein Programm sieht am Beispiel 'CATALOG' so aus:

```
10 PRINT CHR$(4);"CATALOG"
```

Wird dieses Programm durchgeführt, so gibt es den String 'CATALOG' an das Betriebssystem weiter, und dieses zeigt das Inhaltverzeichnis der Diskette im Laufwerk an.
In vielen Programmen wird das Zeichen CONTROL-D häufig gebraucht. Deshalb wird es in eine Textvariable geschrieben. Für diese Textvariable hat sich der Name D$ eingebürgert. Wenn wir also ein Programm schreiben, welches Befehle an das Betriebssystem enthält, so definieren wir ziemlich zu Beginn des Programms den String:

```
10 D$ = CHR$(4)
```

Mit dem nachfolgenden Hinweis

```
10 D$ = CHR$(4) : REM - CONTROL-D
```

können wir diese Definition von D$ erläutern. Alle in diesem Programm vorhandenen Befehle können nun das D$ verwenden und sehen beispielsweise so aus:

```
10 D$ = CHR$(4) : REM - CONTROL-D
20 PRINT D$;"CATALOG"
```

Unter DOS 3.3 wird CONTROL-D nur dann richtig erkannt, wenn zuvor ein RETURN ausgeführt wurde. Daß ist nicht der Fall bei einem PRINT-Statement mit Komma oder Strichpunkt bzw. einem TAB. Wenn in einem (DOS 3.3) Programm eine Anweisung statt an das Betriebssystem an den Bildschirm geleitet wird, so muß in das Programm vor der entsprechenden Stelle durch ein PRINT- oder TAB (0)-Statement ein RETURN eingesetzt werden. Alternativ können Sie auch D$ als CHR$(13) + CHR$(4) definieren. CHR$(13) ist gleichbedeutend mit RETURN.

Einige Betriebssystembefehle sind schon in vorhergehenden Kapiteln beschrieben worden. Deshalb werden sie hier zur Wiederholung nur kurz behandelt:
RUN „Filename" liest ein Programm von der Diskette ein und startet es. Vorhandene Programme und Daten werden gelöscht, wenn sie nicht geschützt sind.
SAVE „Filename" speichert ein vorhandenes Programm unter dem Namen „Filename" auf die Diskette.
LOAD „Filename" liest ein auf der Diskette vorhandenes Programm in den RAM Speicher des Computers, ohne es zu starten. Im RAM vorhandene ungeschützte Programme werden gelöscht.

CATALOG (in ProDOS auch CAT) stellt das Inhaltsverzeichnis der Diskette dar.
EXEC „Filename" interpretiert den Inhalt des Files (Textfile) wie eine Tastatureingabe.
INIT „Filename" formatiert eine Diskette (nur unter DOS)
LOCK „Filename" markiert ein File auf der Diskette als schreibgeschützt
UNLOCK „Filename" hebt den durch LOCK hervorgerufenen Schreibschutz wieder auf.
VERIFY „Filename" prüft die Existenz des Files auf der Diskette.
DELETE „Filename" löscht ein auf der Diskette vorhandenes ungeschütztes File.
RENAME „Filename1",„Filename2" ändert den Namen eines Files auf der Diskette.
MAXFILES „Filezahl" definiert die maximale Anzahl gleichzeitig offener Files und reserviert den notwendigen Speicherplatz für Puffer (nur unter DOS 3.3).
PRINT **FRE(0)** ist ein Applesoft Befehl, der die Anzahl noch freier Speicherplätze in den ersten 64 kB angibt. Gleichzeitig wird die Garbage Collection (entfernen unnötiger Strings) ausgeführt. Ist das angezeigte Ergebnis kleiner als 0, müssen 65536 addiert werden, um die richtige Zahl freier Speicherstellen zu erhalten.

Im Folgenden werden die Befehle behandelt, die in den allgemeinen Kapiteln am Anfang des Buches nicht erwähnt sind:

Die Befehle gelten sowohl für DOS 3.3 als auch für ProDOS, außer sie sind besonders gekennzeichnet.
Die möglichen Parameter sind am Schluß des Abschnitts beschrieben.

CHAIN „Filename" (nur unter ProDOS)
liest ein Basic Programm von der Diskette ein und startet es, löscht dabei das alte Programm im Computer, ohne die Daten (Variablen) zu zerstören. CHAIN ist auch in Integer Basic implementiert.
CREATE „Name" (nur unter ProDOS)
erzeugt ein Directory (oder ein File, falls angegeben) auf der Diskette.
FLUSH „Filename" (nur unter ProDOS)
überträgt den Inhalt des Filepuffers im RAM (I/O Puffer) auf das File, schließt es aber nicht.
FP (nur unter DOS 3.3)
schaltet (von Integer Basic) auf Applesoft Basic. (Bestehendes Programm und Daten werden gelöscht.)

FRE (nur unter ProDOS)
FRE führt die Garbage Collection aus (Beispiel 10 PRINT CHR$(4); "FRE"), dabei werden unnötige Strings entfernt und der freie Speicherbereich entsprechend vergrößert. FRE ersetzt den Applesoftbefehl PRINT FRE(0), der auch eine Garbage Collection durchführt, aber sehr viel langsamer ist.

IN# „Slotnummer"
schaltet die Eingabe auf den angegebenen Kanal. Im Prinzip sind nur IN#0 und IN#2 sinnvoll. Sie schalten die Eingabe auf die Tastatur (IN#0) bzw. auf Slot 2 (IN#2), in dem ein Modem oder ein Akustikkoppler installiert werden kann.

INT (nur unter DOS 3.3)
schaltet (von Applesoft Basic) auf Integer Basic. (Bestehendes Programm und Daten werden gelöscht.)

MON C,I,O (nur unter DOS 3.3)
stellt die Anweisungen an den Diskettencontroller (C), die von der Diskette gelesenen (I), bzw. die an die Diskette augegebenen Daten (O) dar. C,I,O können wahlweise, sowohl einzeln, als auch gleichzeitig verwendet werden.

NOMON (nur unter DOS 3.3)
schaltet den MON Befehl wieder aus.

PREFIX (nur unter ProDOS)
Prefix ist eine Systemvariable, die den Namen der aktuellen Diskette enthält. PREFIX wird vom System verwendet, wenn bei Filenamen kein Volumename oder keine Slotnummer (auch Laufwerksnummer) angegeben ist. Nach dem Booten ist zunächst der Name der Boot-Diskette PREFIX. Der Name kann geändert werden z.B. durch

PREFIX/Prodisk2/Dir3 oder
PREFIX/BP

Dabei bezeichnet Prodisk2 oder BP einen Diskettennamen (Volumename), Dir3 einen Directorynamen. Die Befehle CAT oder CATALOG mit Angabe von Slot und/oder Laufwerksnummer definieren den PREFIX neu.

PREFIX/RAM

definiert die RAM-Disk als aktuelle Disk.

PREFIX ,D1 ,S6

definiert das eingebaute Laufwerk als Prefix.

PREFIX/

löscht den Prefix.

PREFIX

zeigt den aktuellen Prefix an.

PR# „Slotnummer"
setzt den Output auf Slotnummer.

PR#0	aktiviert 40 Zeichendarstellung
PR#1	aktiviert Slot 1 (Drucker)
PR#2	aktiviert Slot 2 (Modem)
PR#3	aktiviert 80 Zeichendarstellung
PR#4	aktiviert den Mauseingang
PR#5	hat keine Bedeutung
PR#6	bootet vom eingebauten Disklaufwerk
PR#7	bootet vom externen Diskettenlaufwerk

STORE „Filename" (nur unter ProDOS)
speichert die verwendeten Variablen unter dem angegebenen Namen auf Diskette.

RESTORE „Filename" (nur unter ProDOS)
liest die mit STORE gespeicherten Variablen eines Programms wieder von der Diskette ein.

– „Filename" (nur unter ProDOS)
liest und startet File von Diskette. Der Befehl führt je nach Filetyp ein RUN, BRUN oder EXEC aus.

Die Befehle können mit **Parametern** versehen werden, die an den Befehl angehängt werden.
Beispiel:

BSAVE /DISK12/REFFILE ,A2048, L500, TBIN, S6, D1

Parameter	Min	Max	DOS	ProDOS
,D Laufwerksnummer	1	2	x	x
,S Slotnummer	1	7	x	x
,A Anfangsadresse	0	65535	x	x
,L Länge	1	65535	x	x
,E Endadresse	1	65535		x
,T Filetyp	siehe unten			x
,F Feldnummer	0	65535		x
,R Recordnummer	0	65535		x
,B Bytenummer	0	Record-1	x	x
,= Zeilennummer	0	65535		x

Die angegebenen Grenzen gelten ebenso wie die untere Darstellung der zulässigen Parameter der Befehle für ProDOS.

Befehl	Parameter								
	,D	,S	,A	,L	,E	,T	,F	,R	,B ,=
APPEND	x	x		x		x			
BLOAD	x	x	x	x	x	x			x
BRUN	x	x	x	x	x				x
SAVE	x	x	x	x	x	x			x
CAT	x	x							
CATALOG	x	x							
CHAIN	x	x							x
CREATE	x	x				x			
CLOSE									
DELETE	x	x							
EXEC	x	x				x	x		
FLUSH									
FRE									
IN#		(x)	(x)						
LOAD	x	x							
LOCK	x	x							
OPEN	x	x		x			x		
POSITION							x	x	
PREFIX	x	x							
PR#		(x)	(x)						
READ					x	x	x		
RENAME	x	x							
RESTORE	x	x							
RUN	x	x						(x)	
SAVE	x	x							
STORE	x	x							
UNLOCK	x	x							
WRITE						x	x	x	
-	x	x							

Bei den in Klammern angegebenen Parametern werden nur die Nummern verwendet. Bei IN# und PR# erfolgt Adreßangabe ohne Komma (z.B. PR# A768), wenn keine Slotnummer angegeben ist.

ProDOS erlaubt folgende Filetypen, die bei manchen Befehlen mit dem Parameter ,T Typ angegeben werden Können.

Beispiel:

CREATE DATPROG,TBAS

Typ	Beschreibung
DIR	Directory
TXT	Textfile
BAS	Applesoftprogramm
VAR	Applesoft Variablenfile
BIN	Binaryfile
REL	Relocatable Code
$F1 bis $F8	Benutzerdefiniertes File
SYS	ProDOS Systemfile

3.12 Der direkte Zugriff zum RAM-Speicher

Befehle: **PEEK, POKE, CALL, HIMEM:, LOMEM:**

Der Apple besitzt 128 kB RAM Speicher. Dieser RAM Speicher ist unterteilt in 2 mal 64 kB. Die Unterteilung ist eine Folge des Prozessors 65C02. Dieser Prozessor kann maximal einen Speicherbreich von 64 kB (entpricht einer Auflösung von 16 Bit) „adressieren" oder verwalten; einfach gesagt, mehr als 65536 (das sind 64 kB) verschiedene Speicherzellen kann er nicht mehr unterscheiden. Um nun 128 kB Speicher benutzen zu können, braucht der Prozessor spezielle Programme, die zwischen diesen beiden 64 kB Teilen hinundher schalten (Bankswitching). Der Prozessor selbst „sieht" immer nur 64 kB von diesen insgesamt 128 kB auf einmal. Für praktisch alle selbstgeschriebenen Programme genügen die ersten 64 kB. Nur wenige Programme brauchen auch den zweiten Teil (z.B. APPLEWORKS).
Das Betriebssystem ProDOS erlaubt die Verwendung der zweiten 64 kB als RAM-Disk (der Speicherbereich wird wie eine Diskette behandelt, arbeitet wegen fehlender Mechanik aber wesentlich schneller). Das Vorgehen zur Verwendung der RAM-Disk ist in Abschnitt 3.22 beschrieben.

In jedem der 65536 Speicherplätze der ersten 64 kB ist eine Zahl zwischen 0 und 255 gespeichert (das gibt 256 verschiedene Möglichkeiten, entspricht 8 Bit, 2 hoch 8 = 256). Alle Programme und Daten liegen also nur in einer Folge von Zahlen zwischen 0 und 255 vor.
Die Speicherzellen sind durchnumeriert von 0 bis 65535. Es gibt eine zweite Numerierungsmethode von -32768 bis 32768 (die zweite Methode bezeichnet die Adressen in einer anderen Reihenfolge als die erste). Von der ersten Numerierung kommt man zur zweiten, indem man die Zahl 65536 (nicht 65535) subtrahiert. Die Adressen 65535 und -1 bezeichnen also dieselbe Speicherzelle. Wundern Sie sich also nicht, wenn negative Adressen vorkommen.

3.12.1 Speicherinhalte lesen, PEEK

Man kann den Inhalt einer bestimmten Speicherzelle aufrufen mit Hilfe des Befehls PEEK(Adresse):

```
PRINT PEEK(35)
24
```

Normalerweise ist in dieser Adresse die Zahl 24 gespeichert. Der Inhalt dieser Speicheradresse gibt den unteren Rand des Bildschirmausschnitts an. Normalerweise ist der ganze Bildschirm aktiviert, hat also 24 aktive Zeilen, die von 1 (oberste Zeile) bis 24 (unterste Zeile) durchnumeriert sind. Aus diesem Grund darf in der Speicherzelle 35 nur ein Wert von 1 bis 24 stehen. Dieser Wert in Zelle 35 muß außerdem größer sein als der von Zelle 34. Der Wert in Zelle 34 gibt den oberen Rand des aktiven Bildschirmausschnitts wieder.

Beim Aufrufen bestimmter Speicherzellen ist Vorsicht geboten. Es kann vorkommen, daß gewisse Funktionen allein dadurch ausgelöst werden, daß eine bestimmte Speicherzelle gelesen wurde (der Inhalt muß nicht einmal auf dem Bildschirm dargestellt werden). Eine solche Funktion wird durch

```
PRINT PEEK(-16336)
```

ausgelöst. Hier wird ein Knacken im Lautsprecher erzeugt. Die in PEEK verwendete Adresse kann auch berechnet werden, z.B. um ganze Speicherbereiche zu untersuchen:

```
FOR I = 768 TO 788 : PRINT PEEK( I ): NEXT I
```

3.12.2. Speicherinhalte ändern, POKE

Speicherzellen kann man nicht nur lesen, sondern natürlich auch einen Wert (0 bis 255) hineinschreiben. Das geschieht mit dem Befehl POKE. Schreiben Sie zunächst den Befehl HOME, um den Cursor in die oberste Zeile zu rücken, und anschließend

POKE 35,15

Durch diesen Befehl wird in Adresse 35 der Wert 15 gespeichert. Der Wert in Adresse 35 stellt den unteren Rand des Bildschirmausschnitts (1 bis 24) dar. Mit dem Befehl CATALOG zum Beispiel können Sie überprüfen, ob tatsächlich nur die obersten 16 Zeilen des Bildschirms aktiviert sind. Mit POKE 35,24 können Sie wieder den ganzen Bildschirm reaktivieren. Wenn Zahlen größer als 24 in Adresse 35 gespeichert werden, können Daten im RAM überschrieben werden.

Wie schon erwähnt, kann durch PEEK( ) eine Funktion ausgelöst werden. Durch den Befehl POKE erfolgt der Aufruf einer solchen Funktion gleich zweimal nacheinander. Bei POKE -16336,0 knackt der Lautsprecher zweimal kurz hintereinander. In einem solchen Fall spielt der dabei abgespeicherte Wert keine Rolle.

Genauso wie bei PEEK kann auch die bei POKE verwendete Adresse der Speicherzelle und auch der zu speichernde Wert berechnet werden:

FOR I = 1 TO 10: POKE 767 + I , 3 * I : NEXT

Wir kennen jetzt zwei Befehle (POKE und PEEK), mit denen wir gezielt Speicheradressen ändern und sichtbarmachen können. In einem BASIC-Programm dienen diese Befehle vor allem für zwei Aufgaben:

1. Gezieltes Ändern oder Ansprechen einer bestimmten Speicherzelle, um eine Funktion auszulösen. z.B:

PEEK -16301 schaltet den Apple auf eine Grafikausgabe um.

PEEK -16303 schaltet den Apple auf Textausgabe um.

2. Speichern und Kontrollieren eines Maschinenprogrammes aus einem BASIC Programm heraus.

Diese Technik ist im Verhältnis zur Programmierung in BASIC relativ aufwendig. Sie wird normalerweise nur angewendet, wenn gewünschte Funktionen in BASIC nicht zu programmieren sind, oder wenn eine sehr hohe Geschwindigkeit erforderlich ist.

Das folgende BASIC-Programm speichert ein Maschinenprogramm, (auch Assemblerprogramm) in den Speicherzellen 768 bis 797 und ruft dieses auf:

```
10 HOME
20 DATA 169, 200, 133, 1, 173, 48, 192, 136, 208, 4, 198, 1, 240,
        9, 202, 208, 246, 166, 0, 232, 134, 0, 244, 0, 240, 3, 76,
        4, 3, 96
30 FOR I = 0 TO 29: READ A: POKE 768 + I, A : NEXT
40 FOR I = 0 TO 4: CALL 768 : NEXT
```

In Zeile 20 sind 30 Zahlen vorhanden, die in Zeile 30 mit dem Befehl POKE in die Speicherzellen 768 und folgende geschrieben werden.
Die mit POKE gespeicherten Zahlen (169, 200, 133, etc.) stellen das Maschinenprogramm dar. Das Programm erzeugt Töne; der Mechanismus eines solchen Maschinenprogramms soll durch den beim letzten POKE abgespeicherten Wert 96 verdeutlicht werden:
Diese Zahl 96 bedeutet in dem Maschinenprogramm den Befehl RTS (ReTurn from Subroutine). Dieser arbeitet analog zum RETURN in BASIC am Ende eines GOSUB-Unterprogrammes. Nach Ausführung des Befehls RTS im Maschinenprogramm erfolgt ein Rücksprung in das aufrufende Programm, hier Zeile 40 des Basic-Programms. Der APPLESOFT-Interpreter fährt mit dem BASIC-Programm fort. In Zeile 40 wird das Maschinenprogramm mit dem Befehl CALL 768 fünf mal aufgerufen.
CALL entpricht dem Befehl GOSUB. Statt wie bei GOSUB in eine bestimmte Zeile zu verzweigen, springt CALL zu der nachstehenden Adresse, führt das Programm aus, welches dort vorhanden ist, und kehrt ins BASIC Programm zurück, wenn es auf den Befehl RTS (96) trifft.

Der Befehl CALL kann auch mit Variablen verwendet werden. Das macht das Programm leichter lesbar. Im obigen Programm ist der Vorteil wegen der geringen Programmlänge nicht bedeutend, der Sinn wird aber klar:

```
35 SOUND = 768
40 FOR I = 0 TO 4: CALL SOUND : NEXT
```

Maschinenroutine aus Basic aufrufen

```
100 POKE 768,169
110 POKE 769,200
120 POKE 770,133
...
...
180 CALL 768
```

verweigt mit CALL ins Maschinen-unterprogramm;

Basic Programm speichert Maschinen-programm,

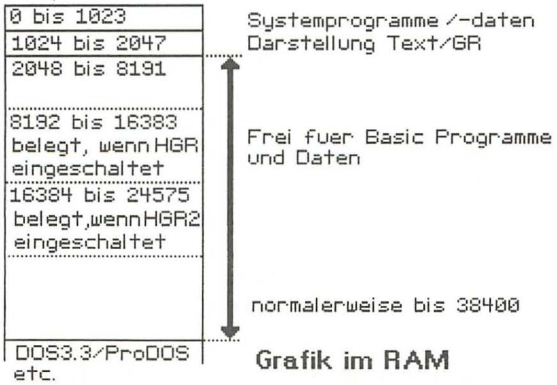
```
$300   A9
$301   C8
$302   85
...    .
       60
```

```
190 ...
200 ...
```

und kehrt nach der Ausfuehrung mit RTS (60) ins aufrufende Basic Programm zurueck.

3.12.3 Speicheraufteilung

Der RAM Speicher des Apple //c ist etwas verwirrend belegt. Er enthält Teile des Betriebssystems, eventuell Programmiersprachen, Programme und Daten. Diese verschiedenen Informationen sind nicht systematisch z.B. von Speicherzelle Null aufwärts vorhanden, sondern über den gesamten Speicherbereich verstreut. Diese Aufteilung entspringt aus der geringeren Speicherkapazität der Vorgängermodelle des Apple //c. Die im Laufe der Zeit schrittweise von 4 kB auf 64 kB vergrößerte Speicherkapazität erlaubte immer weitere zusätzliche



Software. Diese neue Software wurde in die noch freien Bereiche gelegt, so daß schließlich die jetzige Speicheraufteilung entstand.
Programmiert man reines APPLESOFT, also ohne POKE, PEEK, oder CALL, so merkt man von der Speicheraufteilung praktisch nichts. Werden diese Befehle jedoch verwendet, so sollte man einen ungefähren Überblick über die Aufteilung des RAM Speichers haben.

Eine detaillierte Darstellung der Speicheraufteilung findet sich im Anhang.

3.12.4 Hexadezimale Adressen

In der Darstellung der Speicheraufteilung sind die Adressen oder Speicherplätze sowohl dezimal als auch hexadezimal bezeichnet. Die hexadezimale Darstellung ist eine Folge des Dualsystems, mit dem der Computer arbeitet.

Zahlensysteme

Dezimal	Hexadezimal	Binär (4 Bit)
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

Größere Zahlen werden mit mehr Stellen geschrieben; z.B.

Dezimal	Hexadezimal	Binär (8 Bit)
16	10	00010000
17	11	00010001
.		
200	C8	11001000
.		
255	FF	11111111

3.13 Zugriff auf Disketten

Befehle: **APPEND, CLOSE, OPEN, POSITION, READ, WRITE**

Eine Reihe von Basic-Befehlen für den Diskettenzugriff wurden schon in anderen Abschnitten besprochen. Zu diesen Befehlen, die in diesem Abschnitt nicht mehr behandelt werden, gehören SAVE, LOAD, CAT, CATALOG, BLOAD, BSAVE, DELETE, LOCK, UNLOCK, VERIFY und RENAME.
Dieser Abschnitt handelt im besonderen vom Zugriff auf Datenfiles (Textfiles). Die ProDOS-spezifischen Filestrukturen mit benutzerdefinierbaren Filetypen sind im ProDOS Reference Manual beschrieben.
Die Darstellung der Befehle gilt sowohl für die Verwendung mit DOS 3.3, als auch mit ProDOS. Sonderparameter und spezielle Formen der besprochenen Befehle in der ProDOS Version sind in Abschnitt 3.13.6 beschrieben.

Textfiles enthalten im Gegensatz zu Programmfiles nur Daten in Textform. Zu diesen Daten gehören allerdings auch Zahlen, so daß der Name Textfile im engeren Sinn nicht zutrifft.
Die Syntax zur Handhabung von Textfiles ist etwas kompliziert, da das Betriebssystem (sowohl DOS wie auch ProDOS) erst nach der Sprache Applesoft Basic entwickelt wurde. Aus diesem Grund sind in Applesoft Basic keine Befehle für den direkten Zugriff auf Textfiles vorhanden, sondern solche Befehle müssen an das Betriebssystem weitergeleitet werden. Dazu wird vor dem eigentlichen Lese- oder Schreibbefehl ein Kontrollzeichen eingefügt. Das Kontrollzeichen ist CONTROL-D. In den folgenden Beispielen wird es durch die Anweisung CHR$(4) erzeugt.

Beispiel: Lesen von einem Textfile:

```
10 PRINT CHR$(4); "OPEN TEXT.FILE1"      (File öffnen)
20 PRINT CHR$(4); "READ TEXT.FILE1"      (als Lesefile markieren)
30 INPUT A$                              (Daten einlesen)
40 PRINT CHR$(4); "CLOSE TEXT.FILE1"     (File schließen)
```

Zeile 10 öffnet das File mit dem Namen „TEXT.FILE1"; Zeile 20 gibt an, ob vom File gelesen oder ob es beschrieben werden soll. In Zeile 30 wird schließlich die Variable A$ eingelesen und in Zeile 40 das File wieder geschlossen. Die Variable A$ kann nun ausgedruckt, geändert oder sonst weiterverarbeitet werden.

In obigen und den folgenden Programmen werden Strichpunkte verwendet, diese können jedoch auch weggelassen werden. Beispiel:

```
40 PRINT CHR$(4) "CLOSE TEXT.FILE1"
```

Analog zum Lesen vom Textfile geschieht das Schreiben.

Beispiel: Schreiben auf ein Textfile

```
10 PRINT CHR$(4); "OPEN TEXT.FILE1"       (File öffnen)
20 PRINT CHR$(4); "WRITE TEXT.FILE1"      (als Schreibfile
                                           markieren)
30 PRINT "String1"                        (Daten schreiben)
40 PRINT CHR$(4); "CLOSE TEXT.FILE1"      (File schließen)
```

In dem Beispiel wird auf ein File namens TEXT.FILE1 der Text „String1" geschrieben; ist das File schon vorhanden (und nicht mit LOCK schreibgeschützt), so wird der bisherige Inhalt überschrieben; ist das File noch nicht vorhanden, so wird es eröffnet.

Statt „Text" kann ein Textfile auch numerische Daten enthalten:

```
10 PRINT CHR$(4); "OPEN TEXT.FILE2"       (File öffnen)
20 PRINT CHR$(4); "WRITE TEXT.FILE2"      (als Schreibfile
                                           markieren)
30 PRINT 911.01                           (Daten schreiben)
40 PRINT 4711
50 PRINT -1.3E-5
60 PRINT CHR$(4); "CLOSE TEXT.FILE2"      (File schließen)
```

Die Zahlen werden wiederum als numerische Daten eingelesen:

```
10 PRINT CHR$(4); "OPEN TEXT.FILE2"      (File öffnen)
20 PRINT CHR$(4); "READ TEXT.FILE2"      (als Lesefile markieren)
30 INPUT A                               (Daten einlesen)
```

```
40 INPUT B
50 INPUT C
60 PRINT CHR$(4); "CLOSE TEXT.FILE2"     (File schließen)
70 PRINT A,B,C
RUN

911.01          4711          -1.3E-5
```

Der Name Textfile ist auch dann zutreffend, wenn das File mit Zahlen beschrieben wurde, denn:

```
10 PRINT CHR$(4); "OPEN TEXT.FILE2"      (File öffnen)
20 PRINT CHR$(4); "READ TEXT.FILE2"      (als Lesefile markieren)
30 INPUT A1$: INPUT A2$: INPUT A3$       (Daten einlesen)
40 PRINT CHR$(4); "CLOSE TEXT.FILE2"     (File schließen)
50 PRINT A1$,A2$,A3$
RUN

911.01          4711          -1.3E-5
```

Anmerkung: Man beachte die Variablen A1$,A2$,A3$!

Obwohl das File „TEXT.FILE2“ numerisch beschrieben wurde, also Zahlen enthält, können die Zahlen als Strings wieder eingelesen werden! Das File enthält nämlich alle Daten in Form von Text, der je nach Programm einer Textvariablen (Stringvariablen, z.B. A$) oder einer numerischen Variablen (z.B. A) zugeordnet wird.
Die im obigen Beispiel verwendete Variable A1$ enthält „911.01“ als Text, nicht als Zahl! Dieser Text kann jedoch mit dem Befehl VAL() nach dem Einlesen in eine Zahl umgewandelt werden (z.B. A = VAL(A1$)).

Ebenso ist es möglich, von einem Textfile, welches mit Strings beschrieben wurde, numerische Werte einzulesen (natürlich nur dann, wenn die Strings Ziffern zwischen 0 und 9, +,-, E enthalten, z.B. A$= “-911E+4“). Sind in den Strings Buchstaben (außer „E“) oder andere nichtnumerische Zeichen enthalten, so treten Lesefehler auf.

3.13.1 Das „OPEN“-Problem

In den bisher verwendeten Beispielen wurde der Befehl „OPEN Filename“ verwendet. Der OPEN Befehl „öffnet“ eine Datei. Ist die Datei noch nicht vorhanden, so wird sie automatisch angelegt.

Bestehende Dateien brauchen nicht geöffnet werden:

```
20 PRINT CHR$(4); "READ TEXT.FILE2"      (als Lesefile markieren)
30 INPUT A$                              (Daten einlesen)
40 PRINT CHR$(4); "CLOSE TEXT.FILE2"     (File schließen)
```

Das Programm ergibt eine Fehlermeldung, wenn das File noch nicht existiert, funktioniert aber einwandfrei, wenn das File vorhanden ist.
Die Verwendung von „OPEN Filename" vermeidet eine solche Fehlermeldung, hat aber den Nachteil, daß „leere" Files erzeugt werden können. Die Erzeugung eines „leeren" Files geschieht logischerweise nur beim Versuch, ein nicht vorhandenes Files zum Lesen zu öffnen. Durch eine Prüfung der Länge der eingelesenen Information können diese Fälle vom Programm festgestellt werden (Länge = 0), danach das erzeugte „leere" File gelöscht und der Benutzer auf den Fehlversuch aufmerksam gemacht werden.

Wird auf ein bereits existierendes File mit dem WRITE Befehl geschrieben, so wird der vom alten File belegte Platz auf der Diskette auch für das neue File verwendet. Ist das neue File größer als das alte, so wird (soweit vorhanden) zusätzlicher Platz auf der Diskette belegt. Ist umgekehrt das neue File kleiner als das alte, so bleibt trotzdem der gesamte, vom alten File belegte Platz auch für das neue reserviert. Um unnötige Platzverschwendung auf der Diskette durch reservierten, aber nicht benutzten Speicherplatz zu vermeiden, wird die OPEN-CLOSE-DELETE-OPEN Kombination verwendet.

```
10 PRINT CHR$(4); "OPEN Filename"
20 PRINT CHR$(4); "CLOSE Filename"
30 PRINT CHR$(4); "DELETE Filename"
40 PRINT CHR$(4); "OPEN Filename"
...
```

Das erste „OPEN" und „CLOSE" verhindert eine (eventuelle) Fehlermeldung bei der Ausführung des „DELETE'-Befehl, falls das File nicht existiert. „DELETE" löscht das alte File und gibt den gesamten von ihm belegten Platz frei. Das mit dem zweiten „OPEN" eröffnete neue File belegt nur den Platz, den es wirklich benötigt. Unter dem Betriebssystem DOS 3.3 kann der „CLOSE" Befehl in diesem Fall weggelassen werden.
Anmerkung: Die etwas langwierige Befehlskombination OPEN-CLOSE-DELETE-OPEN kann mehrmaliger Verwendung im Programm in einem String untergebracht werden, siehe unten.

Außer der OPEN-CLOSE-DELETE-OPEN Kombination ist noch die (OPEN-WRITE-CLOSE)-DELETE-RENAME Kombination möglich. Dabei erfolgt

zunächst die normale Speicherung der Daten unter einem anderen Filenamen. Anschließend wird das alte (eigentlich zu überschreibende) File mit DELETE gelöscht und das neue File mit RENAME umbenannt. Bei dieser Methode werden erst dann alte Daten gelöscht, wenn sichergestellt ist, daß die neuen Daten gespeichert sind. Allerdings läßt sich mit der Methode die ohnehin schon geringe Speicherkapazität der Apple Disketten nicht voll ausnutzen, was sich besonders bei großen Files bemerkbar macht.

Anmerkung für ProDOS-Anwender:
Für jedes File muß bei den Anweisungen WRITE, READ, CLOSE etc. genau die Filebezeichnung verwendet werden, die auch in der OPEN Anweisung benutzt wurde. Die Definition eines Prefix nach der OPEN Anweisung hat für das (die) geöffnete(n) File(s) keine Wirkung; der Filename wird nur dann erkannt, wenn er wie in der OPEN Anweisung verwendet wird.

```
10 PRINT CHR$(4);"OPEN /DISK2/TEST"          (File öffnen)
20 PRINT CHR$(4);"PREFIX /DISK2/"            (Prefix setzen)
30 PRINT CHR$(4);"READ TEST"                 (Neuer Filename)
....
RUN
```

FILE NOT OPEN

3.13.2 Files schließen, CLOSE und MAXFILES

Der Befehl „CLOSE Filename“ schließt das File „Filename“. „CLOSE“ allein schließt alle offenen Files.
Unter dem Betriebssystem DOS 3.3 entsteht primär keine Fehlermeldung, wenn Files offen bleiben. Da ein offenes File aber RAM-Speicherplatz belegt, sollte es bei Nichtgebrauch geschlossen werden. Sind unter DOS 3.3 zu viele Files (mehr als 3) gleichzeitig offen, so erscheint die Fehlermeldung „NO BUFFERS AVAILABLE“. Unter DOS 3.3 kann mit dem Befehl MAXFILES + Zahl die Anzahl der maximal gleichzeitig offenen Files angegeben werden:

MAXFILES 4

reserviert ab Speicherplatz 37805 (HIMEM) für die angegebene Filezahl je 595 Byte als Datenpuffer. Nach dem Booten wird automatisch Pufferplatz für 3 Files reserviert (ab Speicherplatz 38400 ($9600), dem normalen HIMEM Wert). Durch Ändern der Pufferzahl mit MAXFILES wird HIMEM nach unten oder oben verschoben. Somit ändert sich der für Programmdaten vorhandene Spei-

cherbereich (unterhalb von HIMEM). Im Bedarfsfall muß MAXFILES daher vor der Definition von Textvariablen (Strings) gesetzt werden.
Auch Befehle wie CATALOG oder LOAD benötigen einen Filepuffer, der nach Beendigung des Befehls wieder freigegeben wird.

ProDOS reserviert zunächst nur einen Puffer (1 kByte), und richtet weitere Puffer bei Bedarf selbsttätig ein.

3.13.2.3 Zwischenaktivierung des Basic-Interpreters

Häufig ist es wünschenswert, bei geöffnetem Output-File Meldungen für den Benutzer auszugeben. Um diese Meldungen auf den Bildschirm zu bringen und nicht in das File, muß vor der Meldung ein „nacktes" CONTROL-D (CHR$(4)) ausgegeben werden.
Beispiel:

```
10 PRINT CHR$(4); "OPEN TEXT.FILE1"
20 PRINT CHR$(4); "WRITE TEXT.FILE1"
30 PRINT A$
40 PRINT CHR$(4)
50 PRINT "A$ ist gespeichert"
60 PRINT CHR$(4); "WRITE TEXT.FILE1"
70 PRINT B$
80 PRINT CHR$(4); "CLOSE TEXT.FILE1"
```

Durch das CHR$(4) in Zeile 40 wird der WRITE Befehl neutralisiert und in Zeile 50 der Bildschirm angesprochen.
Das gleiche Problem entsteht beim Lesen von einem Textfile und gleichzeitiger Fragestellung am Bildschirm:

```
10 PRINT CHR$(4); "OPEN TEXT.FILE1"
20 PRINT CHR$(4); "READ TEXT.FILE1"
30 INPUT A$
40 PRINT CHR$(4)
50 INPUT "String eingelesen, weitermachen ?";X$
60 PRINT CHR$(4); "READ TEXT.FILE1"
70 INPUT B$
80 PRINT CHR$(4); "CLOSE TEXT.FILE1"
```

Das Vorgehen gilt für alle Betriebssystemanweisungen; die Ausgabe des nackten CHR$(4) ohne Anhang aktiviert den Applesoft-Interpreter.

3.13.3 Sequentielle Textfiles

In den bisherigen Beispielen wurde die einfachste Form von Textfiles, sogenannte sequentielle Textfiles verwendet. Sequentielle Textfiles enthalten die Daten in der Reihenfolge, in der sie auf das File geschrieben wurden. Neben sequentiellen Textfiles gibt es sogenannte RANDOM ACCESS Files (RANDOM ACCESS = wahlloser, beliebiger Zugriff), die direkten Zugriff auf eine bestimmte Information im File ermöglichen. Die Handhabung dieser Files ist unten beschrieben.

Sequentielle Files enthalten (in der Regel) unterschiedlich lange Einträge, die nur streng in Reihenfolge gelesen oder beschrieben werden können.

RANDOM ACCESS Files enthalten nur gleich lange (durchnummerierte) Einträge, auf die beliebig zugegriffen werden kann.

Aufbau eines sequentiellen Textfiles; verschieden lange Eintraege sind durch Return (RET) getrennt.

Aufbau eines RANDOM-ACCESS-Files; gleichlange Eintraege (Records) sind durch Returns getrennt. Die Records sind nummeriert, und koennen direkt angesprungen werden.

Die nachfolgenden Abschnitte beschreiben die Vorgehensweise bei sequentiellen Files, RANDOM ACCESS Files werden in Abschnitt 3.13.4 behandelt.

3.13.3.1 Records, Felder und Bytes

Bei den oben dargestellten Beispielen wurden PRINT bzw. INPUT Statements durch Doppelpunkte getrennt, bzw. in verschiedene Zeilen geschrieben:

```
...
30 PRINT "HALLI": PRINT "HALLO"          (Schreiben auf File)
...                                      bzw.

...
30 INPUT A$:INPUT B$                     (Lesen von File)
...
```

In diesem Fall sind die einzelnen Strings (oder Zahlen) auf der Diskette durch RETURNs (CONTROL-M, CHR$(13)) getrennt. Sie werden entsprechend diesem Trennzeichen auch wieder eingelesen.

Anmerkung:
Die Kontrollzeichen CONTROL-@ (ASCII 0), CONTROL-D (ASCII 4) und CONTROL-M (ASCII 13) sollten nicht als Text auf Diskette gespeichert werden. Diese Zeichen dienen dem Basic-Interpreter als Kontrollzeichen für das Betriebsystem oder als Trennzeichen auf dem File. Die Verwendung dieser Zeichen als Text kann daher leicht zu Schwierigkeiten führen.

Die Trennzeichen (RETURN) auf dem File werden unterdrückt, wenn der PRINT Befehl mit einem Strichpunkt endet:

```
...
30 PRINT "HALLI";:PRINT "HALLO";         (Schreiben auf File
...                                      ohne Trennzeichen)
```

Beim Wiedereinlesen hängen die beiden Strings zusammen:

```
...
30 INPUT A$                              (Lesen von File)
40 PRINT A$                              (Ausgabe auf Bildschirm)
...
RUN

HALLIHALLO
```

Durch die Verwendung des PRINT Befehls mit abschließendem Strichpunkt werden die Daten auf dem File als ein einziger großer String abgelegt. Wird dieser Riesenstring länger als 239 Zeichen, so kann er nicht mehr durch ein

INPUT Statement eingelesen werden, da die zum INPUT Statement gehörige Variable (z.B. A$) maximal einen 239 Zeichen langen String enthalten kann. Werden mehrere INPUT Statements nacheinander verwendet, so liest das erste 239 Zeichen ein, das nächste liest weitere 239 Zeichen usw. Die sinnvolle Aufteilung der Filedaten muß anschließend durch ein Benutzerprogramm erfolgen.
Beispiel:

```
10  PRINT  CHR$ (4);"OPENRIESENSTRING"
20  PRINT  CHR$ (4);"READRIESENSTRING"
30  FOR I = 1 TO 100: INPUT A$: PRINT I,A$: NEXT I
40  PRINT  CHR$ (4);"CLOSE"
50  REM Vorsicht! Der INPUT-Befehl von Strings länger
60  REM als 239 Zeichen arbeitet unter DOS nicht einwandfrei.
```

Das Programm liest einen Riesenstring im File RIESENSTRING ein. Dieses File wurde mit dem nachfolgend (im Abschnitt GET) beschriebenen Programm erzeugt.

Datenabschnitte zwischen zwei RETURN Zeichen (CHR$(13)) werden Record genannt. Ein ganzer Record wird durch eine INPUT Anweisung eingelesen, wenn er nicht unterteilt ist.

INPUT A$ liest einen nicht unterteilten Record ein.

Mit Kommas kann ein Record in Felder unterteilt werden. Die PRINT Anweisung dafür sieht wie folgt aus:

PRINT A$,B$,C$ erzeugt einen (aus drei Feldern bestehenden Record)

Die folgenden Statements sind Teile von Programmen für den Filezugriff. Die in Klammern angegebenen Zahlen entprechen einem Byte (z.B. (13) = RETURN). Eingefügte Freizeichen dienen nur der besseren Darstellung, sie sind auf dem File nicht vorhanden:

PRINT "AAAA": PRINT "BBBB": PRINT "CCCC": PRINT "DDDD" ...

erzeugt folgende Daten auf dem File:

AAAA (13) BBBB (13) CCCC (13) DDDD (13) ....

Das File enthält drei Records, bestehend aus je vier Bytes, getrennt durch RETURNs. Das Einlesen der Daten eines solchen Files wurde bereits beschrieben.

Dagegen bewirkt die Anweisung

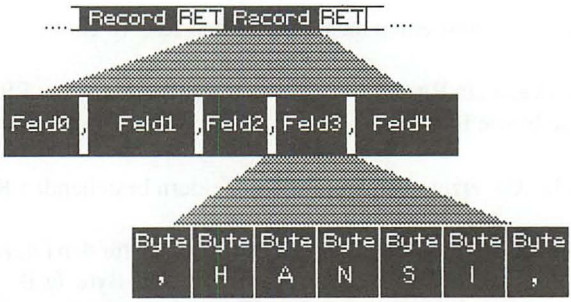
PRINT "AAAA","BBBB","CCCC": PRINT "DDDD","EEEE","FFFF":
...

die Bytefolge:

AAAA , BBBB , CCCC (13) DDDD , EEEE , FFFF (13) ...

Das File enthält zwei Records (getrennt durch RETURNs) bestehend aus je drei Feldern (zu je vier Bytes). Die Felder des Records sind durch Kommas getrennt.

Anmerkung:
Entgegen der Interpretation eines PRINT Statements mit Komma für den Bildschirm, welches eine Tabulatorwirkung hat, wird bei einem PRINT-Statement mit Komma für ein Textfile tatsächlich ein Komma auf das File geschrieben. Das Komma dient zur Trennung der Felder eines Records auf dem File.



Aufbau eines Eintrags (Record) aus Feldern, (getrennt durch Kommas) und Bytes.

Wird von dem zuletzt erzeugten File gelesen, so ergeben sich mindestens 5 signifikant unterschiedliche Möglichkeiten:

```
INPUT A1$: INPUT A2$ : PRINT A1$: PRINT A2$
AAAA
DDDD
```

oder

```
INPUT A1$,A2$ : PRINT A1$: PRINT A2$
AAAA
BBBB
```

oder

```
INPUT A1$,A2$: INPUT A3$,A4$ : PRINT A1$: PRINT A2$
: PRINT A3$: PRINT A4$
AAAA
BBBB
DDDD
EEEE
```

oder

```
INPUT A1$,A2$,A3$ : PRINT A1$: PRINT A2$: PRINT A3$
AAAA
BBBB
CCCC
```

oder

```
INPUT A1,A2$,A3$,A4$ : PRINT A1$: PRINT A2$: PRINT A3$
: PRINT A4$
AAAA
BBBB
CCCC
DDDD
```

Jedes INPUT-Statement liest von der aktuellen Postition bis zum nächsten Trennzeichen (Komma oder Return). Ein INPUT-Statement ohne Komma (INPUT A$) setzt die Position für den nächsten Lesevorgang auf das folgende RETURN. Ein INPUT-Statement mit Komma (INPUT A$,) setzt die Position für den nächsten Lesevorgang auf das nächste Trennzeichen, egal ob RETURN oder Komma.

3.13.3.2 Daten einlesen mit GET

Eine Möglichkeit, nicht getrennte Daten (Riesenstrings) von der Diskette einzulesen, bietet der GET Befehl. Ein File, welches mit

```
10  PRINT  CHR$ (4);"OPENRIESENSTRING"
20  PRINT  CHR$ (4);"WRITERIESENSTRING"
30  FOR I = 1 TO 100: PRINT "HALLO";I;: NEXT I
40  PRINT  CHR$(13); CHR$ (4);"CLOSE"
```

beschrieben wurde, wird unten mit dem GET Befehl gelesen:

```
10  PRINT  CHR$ (4);"OPENRIESENSTRING"
20  PRINT  CHR$ (4);"READRIESENSTRING"
30  FOR I = 1 TO 1000: GET A$: PRINT CHR$(4);: PRINT A$;: NEXT I

40  PRINT  CHR$ (4);"CLOSE"
```

Wie sich durch einen Test mit größeren Files leicht feststellen läßt, arbeitet die Leseroutine mit dem GET Befehl sehr langsam; er wird deshalb nur in Ausnahmefällen verwendet.

Mit dem GET Befehl können Files unbekannter Struktur (ungekannte Recordlänge oder Felderzahl pro Record) inklusive der Trennzeichen eingelesen werden. Die einzelnen Felderinhalte werden dann mit den folgenden Befehlen separiert:

```
10 FOR I=1 TO N
20 GET X$: IF X$ <> "," AND X$ <> CHR$(13) THEN
   A$(I)= A$(I) + X$ : GOTO 20
30 NEXT I
```

3.13.3.3 END OF DATA Fehler

In den bisher verwendeten Programmen wurden nur Daten von Files eingelesen, die dort auch vorhanden waren. In der Praxis tritt aber häufig der Fall auf, daß die Filelänge unbekannt ist. Wird dann versucht, über das Fileende hinaus zu lesen, so erscheint vom Betriebssystem die Fehlermeldung

END OF DATA (keine Daten mehr vorhanden)

Bei ProDOS folgt zusätzlich die Meldung

File(s) still open (File ist (sind) noch offen)

Das eigentlich Lästige an diesem Verhalten ist der Programmabbruch. Mit Hilfe einer ONERR Routine läßt sich das umgehen:

```
5  ONERR  GOTO 100
10  PRINT  CHR$ (4);"OPENRIESENSTRING"
20  PRINT  CHR$ (4);"READRIESENSTRING"
30  FOR I = 1 TO 800: GET A$:PRINT CHR$(4);: PRINT I,A$:
     PRINT A$;: NEXT I
40  PRINT  CHR$ (4);"CLOSE"
50  END
100 REM
          Beginn der Fehlerroutine

110  IF PEEK(222)<> 5 THEN PRINT "Fehler Nr.";PEEK(222):
      POKE 216,0: STOP: REM kein END OF DATA Fehler,
      andere Fehlerbehandlung einfügen.
120  PRINT  CHR$ (4)"CLOSE": PRINT "File ist geschlossen"
130  REM
           Ende der Fehlerroutine

140  POKE 216,0:REM Fehlerroutine ausschalten
150  GOTO 50
```

Das Programm liest vom Textfile RIESENSTRING, welches mit dem Programm im vorhergehenden Abschnitt erzeugt wurde.
Vor der Leseroutine wird die ONERR Routine initialisiert (Zeile 5). Die eigentliche Fehlerbehandlung erfolgt ab Zeile 100, wo zunächst die Fehlerursache geprüft wird. Speicherzelle 222 enthält den Fehlercode (siehe Anhang). Bei Fehlercode 5 (END OF DATA) wird in Zeile 110 das offene File geschlossen und die Maßnahme bestätigt. In Zeile 120 wird mit POKE 216,0 die Fehlerroutine abgeschaltet, damit sie bei einem eventuellen späteren Fehler nicht mehr angesprungen wird.

3.13.3.4 Daten anfügen, APPEND

APPEND öffnet ein File analog zu OPEN, aber mit zwei Unterschieden:
APPEND setzt den Zeiger für den nächsten Schreibvorgang auf das Fileende. APPEND dient daher zum Anfügen neuer Informationen an ein File. Lesebefehle nach APPEND verursachen daher einen „END OF DATA“ Fehler.
APPEND kann keine Files erzeugen. Ist das angegebene File nicht vorhanden, so erscheint die Fehlermeldung „FILE NOT FOUND“.

Die Syntax von APPEND ist genau gleich wie die von OPEN.

3.13.3.5 Zugriff auf einen bestimmten Eintrag, POSITION

Der POSITION Befehl verändert den Zeiger auf den aktuellen Datensatz.

```
10 PRINT CHR$(4);"OPEN FILE1"
20 PRINT CHR$(4);"POSITION FILE1,R10"
30 PRINT CHR$(4);"READ FILE1"
40 INPUT A$
50 PRINT CHR$(4);"CLOSE FILE1"
```

Das Programm öffnet ein File mit OPEN. Damit ist der erste Datensatz (Nr.0) aktuell. Mit POSITION wird nun deA Zeiger um 10 verschoben. Damit wird in Zeile 40 vom 11. Datensatz (Nr.10) gelesen. Durch weitere POSITION Befehle kann der aktuelle Datensatz wieder geändert werden. Nach jedem Position ist wieder ein READ oder WRITE Befehl vor INPUT oder PRINT auszuführen.

3.13.4 RANDOM ACCESS Files (mit direktem Datenzugriff)

Der Aufbau von RANDOM ACCESS Files (oder Random Files) wurde schon oben gestreift. RANDOM ACCESS bedeutet soviel wie frei wählbarer, beliebiger Zugriff auf eine Datensatz, auch als „Record" bezeichnet.

Beispiel:

```
10 PRINT CHR$(4);"OPEN RANDAC,L20"      (File öffnen, dabei
                                         Recordlänge angeben)
20 FOR I=1 TO 100
30 PRINT CHR$(4);"WRITE RANDAC,R";I     (Schreiben, dabei
                                         Recordnr. angeben)
40 PRINT "Textstring Randac"
50 NEXT I
60 PRINT CHR$(4);"CLOSE RANDAC"          (File schließen)
```

In Zeile 10 wurde das Random-File geöffnet, gleichzeitig die Länge eines Records auf 20 gesetzt (mit L20, durch Komma getrennt vom Filenamen). Zwischen Zeile 20 und 50 wiederholt sich 100 mal die gleiche Prozedur: Schreiben mit dem WRITE-Befehl plus Filenamen und durch Komma getrennter Angabe des zu beschreibenden Records (z.B. R12). Anschließend folgt der Text, der auf den angegebenen Record geschrieben werden soll.

Der Lesevorgang geht analog:

```
10 PRINT CHR$(4);"OPEN RANDAC,L20"          (File öffnen)
20 PRINT CHR$(4);"READ RANDAC,R77"          (Lesen)
30 INPUT A$
40 PRINT CHR$(4);"CLOSE RANDAC"             (File schließen)
50 PRINT A$
RUN
```

Textstring Randac

Vom File RANDAC wurde nun direkt der String Nr. 77 eingelesen (Strenggenommen der 78. String. Das File beginnt bei String 0, String 0 wird jedoch in der Regel nicht für Daten verwendet. Es ist nützlich, in String 0 die Länge des Files zu speichern, d.h. die Anzahl der Records)

Unterschiede von Random Files zu sequentiellen Files sind folgende:

RANDOM ACCESS Files

- erlauben direkten Zugriff auf eine bestimmte Zeichenkette (Record)
- bedingen die Definition der Recordlänge (alle Records sind gleich lang)
- benötigen in der Regel mehr Platz als sequentielle Files
- sind beim Zugriff auf wenige Records an verschiedenen Stellen im File schneller als sequentielle Files
- sind beim Zugriff auf alle Records (vom Fileanfang an) langsamer als sequentielle Files

Das soeben Gesagte gilt für Random-Files auf Diskette. Bei Verwendung der RAM-Disk (zweite 64 kB Bank unter ProDOS) sind andere Maßstäbe anzusetzen (siehe Abschnitt über die RAM Disk).

Random Files benötigen mehr Platz auf der Diskette als sequentielle Files, denn jeder Record belegt die definierte Anzahl Bytes, unabhängig davon, ob er vollgeschrieben wurde oder nicht.
Eine Möglichkeit, den Speicherbedarf etwas einzuschränken, ist die blockweise Speicherung von Daten:

3.13.4.1 Beispiel für blockweise Speicherung

Von einer Firma sollen auf Diskette folgende Daten gespeichert werden:

Firmenname	ca. 16 Zeichen
Straße	ca. 15 Zeichen
Postleitzahl	4 Zeichen
Ort	ca. 15 Zeichen
Telefonnummer	ca. 10 Zeichen
TOTAL	60 Zeichen

Die blockweise Speicherung sieht in diesem Fall eine Recordlänge von ca. 60 bis 65 Zeichen vor. Alle Angaben einer Firma werden in einen Record geschrieben. Die jeweiligen Teilinformationen (Firma, Straße etc.) werden voneinander durch ein Kontrollzeichen getrennt.
Der Vorteil dieses Systems ist folgender:
Ist eine Teilinformation (z.B Firmenname) länger als vorgesehen, so kann dies durch andere Informationen (z.B. Ort), die kürzer sind als vorgesehen, wieder ausgeglichen werden. Dadurch läßt sich der Gesamtspeicherbedarf eines Records besser ausnützen.
Die relevanten Teile des obigen Programmbeispiels sehen so aus:

```
10 INPUT "Firmenname  :";F$
20 INPUT "Straße      :";S$
30 INPUT "Postleitzahl:";P$
40 INPUT "Ort         :";O$
50 INPUT "Telefon     :";T$
....

100 PRINT CHR$(4);"OPEN FIRMEN,L60"
110 PRINT CHR$(4);"WRITE FIRMEN,R";RN
120 PRINT F$","S$","P$","O$","T$
130 PRINT CHR$(4);"CLOSE FIRMEN"
....
```

Ein Record des Files FIRMEN sieht dann beispielsweise so aus:

Müller,Holzweg 11,8150,Holzkirchen/Oberbayern,08024/1234

Obwohl die Ortsbezeichnung 'Holzkirchen/Oberbayern' viel zu lang ist (nur 15 Zeichen sind dafür reserviert), besteht der Gesamtstring nur aus 56 Zeichen. Die Kommas im PRINT Statement in Zeile 120 werden mit auf das File geschrieben.

Das Einlesen geschieht analog:

```
100 PRINT CHR$(4);"OPEN FIRMEN,L60"
110 PRINT CHR$(4);"READ FIRMEN,R";RN
120 INPUT F$,S$,P$,O$,T$
130 PRINT CHR$(4);"CLOSE FIRMEN"
....
210 PRINT "Firmenname  :";F$
220 PRINT "Straße      :";S$
230 PRINT "Postleitzahl:";P$
240 PRINT "Ort         :";O$
250 PRINT "Telefon     :";T$
....
```

3.13.4.2 Feld- und Byteweiser Datenzugriff

Bisher haben wir auf einen bestimmten Record (hier Nr.15) zugegriffen mit dem Befehl

```
...
100 PRINT CHR$(4);"READ RANDAC,R15"
110 INPUT A$: PRINT A$
...
RUN
```

Textstring Randac

...

DOS und ProDOS erlauben auch den Zugriff auf ein bestimmtes Byte eines Records:

```
...
100 PRINT CHR$(4);"READ RANDAC,R15,B2"
110 INPUT A$: PRINT A$
...
RUN
```

xtstring Randac

...

Die verwendete Bezeichung „B Nummer", durch Komma von der Recordnummer getrennt, gibt die Bytenummer des Records an. B2 bezeichnet das 3. Byte, weil die Bytes eines Record von 0 an durchnumeriert sind.
Die Definition eines bestimmten Bytes ist auch in der Anweisung POSITION möglich.

3.13.5 Abkürzungen

Die teilweise recht umständliche Schreibweise der Befehle für den Diskettenzugriff können durch Strings abgekürzt werden. Die Maßnahme empfiehlt sich nur dann, wenn sie im Programm an mehreren Stellen verwendet werden:

Beispiel:

```
10 O$ = CHR$(4) + "OPEN": C$ = CHR$(4) + "CLOSE": W$ = CHR$(4)
   + "WRITE": R$ = CHR$(4) + "READ": A$ = CHR$(4) + "APPEND" :
   P$ = CHR$(4) + "POSITON"
...

90 F$="TESTFILE.2"
100 PRINT O$;F$                                    (OPEN)
110 PRINT W$;F$                                    (WRITE)
120 FOR I=1 TO 10 : PRINT X$(I) : NEXT I
130 PRINT C$;F$                                    (CLOSE)
```

Wie man sieht, verschlechtert sich die Lesbarkeit des Programms. Daher ist eine weitere Abstrahierung wie

```
G$=CHR$(4)+"OPEN TESTFILE"+CHR$(13)+CHR$(4)+"POSITION TESTFI-
LE,R44"+CHR$(13)+CHR$(4)+"WRITE TESTFILE"
```

zwar möglich, aber nicht immer sinnvoll.

3.13.6 Filezugriff unter ProDOS

Außer der höheren Geschwindigkeit bietet der Filezugriff mit ProDOS mehr Möglichkeiten als mit DOS 3.3.

3.13.6.1 Filepuffer leeren, FLUSH

Der Befehl FLUSH wirkt ähnlich wie CLOSE. FLUSH schreibt den Inhalt des Filepuffers (Zwischenspeicher für Filedaten) auf das File, schließt dieses jedoch nicht. Damit wird der eventuelle Verlust von Daten vermieden.

FLUSH Superfile schreibt den Filepuffer auf das File

FLUSH kann wie CLOSE mit und ohne Filenamen verwendet werden. FLUSH ohne Filenamen leert alle vorhandenen Filepuffer. FLUSH ist nur beim Schreiben auf Files sinnvoll.

3.13.6.2 Feldweiser Datenzugriff

ProDOS erlaubt den feldweisen Datenzugriff. Felder sind Teile eines Records, die durch Kommas getrennt sind. Bei den Befehlen READ, WRITE und POSITION ist dafür die optionale Angabe des Feldes möglich:

PRINT CHR$(4); "WRITE FILE, F3"

Das Feld wird nach dem Buchtaben F, vom Filenamen getrennt durch Komma, angegeben.

PRINT CHR$(4); "WRITE FILE, R4, F3, B2"

setzt die aktuelle Schreibposition auf Record 4, Feld 3 und Byte 2.

3.13.6.3 Angabe des Filetyps

ProDOS kennt mehr Filetypen als DOS 3.3. Gleichzeitig kann bei den Befehlen CREATE, OPEN und APPEND der Filetyp angegeben werden (gilt übrigens auch für BLOAD und BSAVE). Dazu wird hinter dem Befehl, getrennt durch Komma und einleitendes T, der Typ angegeben.

Beispiel:

CREATE NEUFILE, TBAS erzeugt Basic-File

Die möglichen Filetypen unter ProDOS sind:

Bezeichnung	Filetyp
DIR	Directory
TXT	Textfile
BAS	Basicprogramm
VAR	Variablenfile
BIN	Binärfile
REL	Reloc. Code
$F1 bis $F8	Benutzerdefiniertes File
SYS	ProDOS Systemfile

3.14 Grafik

Befehle: **GR, HGR, HGR2, COLOR=, HCOLOR=, HLIN, VLIN, HPLOT, DRAW, SCALE=, ROT=, XDRAW, SCRN, TEXT**

Der Abschnitt Grafik behandelt vor allem die hochauflösende Grafik (280 x 192 Punkte). Die niedrigauflösenden Grafiken werden der geringeren Bedeutung wegen weniger ausführlich dargestellt. Für doppelthochauflösende Grafik (560 x 192 Punkte) stellt der Apple //c wenig Zeichnungshilfen zur Verfügung. Der Benutzer sollte dafür eines der im Handel erhältlichen Programmpakete benutzen.

Der Apple //c kennt vier verschiedene Grafikmodi:

40 x 48 Punkte niedrigauflösende Grafik mit 16 Farben
80 x 48 Punkte niedrigauflösende Grafik mit 16 Farben
280 x 192 Punkte hochauflösende Grafik mit 6 Farben
560 x 192 Punkte hochauflösende Grafik mit 6 Farben

Die Grafiken belegen je nach Auflösung einen Teil des RAM-Speichers des Apple //c. Niedrigauflösende Grafik (40 x 48) belegt 1 kB, hochauflösende Grafik 280 x 192 belegt 8 kB. Das Thema Grafik und Speicherplatz wird in Abschnitt 3.14.2.3 behandelt.

3.14.1 Niedrigauflösende Grafik, GR

3.14.1.1 40 x 40 (40 x 48) Bildpunkte

Die niedrigauflösende Grafik wird aktiviert mit dem Befehl GR:

GR schaltet niedrigauflösende Grafik ein

Normalerweise werden 40 x 40 Bildpunkte plus vier Zeilen für Text am unteren Bildrand dargestellt (die für Kommentare oder Fragen sehr nützlich sind).
Vor der ersten Zeichnung muß die Farbe gewählt werden. Wird sie nicht gewählt, so ist die Farbnummer gleich 0, d.h. Farbe schwarz (und Hintergrund schwarz). Zeichnungen sind dann nicht sichtbar.

COLOR = 15 schaltet auf Farbe weiß

Der wichtigste Zeichenbefehl ist PLOT (deutsch: darstellen, zeichnen):

PLOT 3,9 zeichnet einen Punkt in Spalte 3, Zeile 9

Der Befehl zeichnet einen (hier weißen) Punkt in Spalte 3 und Reihe 9. Wie in allen Grafiken ist der Punkt 0,0 (der Ursprung) links oben, wie bei HOME im Textmodus. Die horizontale Achse (nach rechts) wird als X Achse bezeichnet, die vertikale Achse (nach unten) als Y Achse.
Die Argumente von PLOT dürfen zwischen 0 und 39 (erste Zahl) bzw. zwischen 0 und 47 (zweite Zahl) liegen.
Ein Strich entsteht aus einer Reihe von berechneten PLOT-Punkten:

FOR I=4 TO 28: PLOT I,I/2: NEXT Linie von 4,2 bis 28,14

Mit den bisher erwähnten drei Befehlen können schon Bilder gezeichnet werden. Das nachfolgende Programm gibt den Verlauf einer Sinusfunktion wieder:

```
10 GR: HOME
20 COLOR = 15
30 FOR I=0 TO 39
40 PLOT I , ( SIN (I / 4) * 10 ) + 20
50 NEXT I
60 VTAB 21 : PRINT " SINUSFUNKTION "
70 END
```

SINUSFUNKTION

Die Befehle HLIN und VLIN zeichnen horizontale bzw. vertikale Linien:

HLIN 4,30 AT 12 zeichnet horizontale Linie von Spalte 4 bis 30 in Zeile 12

VLIN 12,20 AT 4 zeichnet vertikale Linie von Zeile 12 bis 20 in Spalte 4

Der Befehl SCRN ermittelt die Farbe eines Punktes der Grafik.

PRINT **SCRN**(3,9)
15 gibt den Farbwert des Punktes 3,9 an.

Die Farbe (in der anschließend gezeichnet wird) wird mit dem erwähnten COLOR Befehl auf Werte zwischen 0 und 15 gesetzt. Die COLOR-Werte entsprechen den folgenden Farben:

COLOR Wert	Farbe	COLOR Wert	Farbe
0	schwarz	8	braun
1	rot	9	orange
2	dunkelblau	10	grau
3	purpur	11	rosa
4	dunkelgrün	12	hellgrün
5	grau	13	gelb
6	blau	14	aquamarinblau
7	hellblau	15	weiß

Die Farben werden natürlich nur mit einem Farbbildschirm sichtbar, auf einfarbigen Monitoren entstehen unterschiedliche Grautöne.

Ähnlich wie bei hochauflösender Grafik beschrieben, kann auch in niedrig auflösender Grafik die Maus verwendet werden. Ein entprechendes Programm ist im Handbuch zur Maus dargestellt.

Um statt mit den bisher dargestellten 40 mal 40 Punkten (mit 4 Zeilen Text) auch mit 40 mal 48 Punkten ohne Textdarstellung zu arbeiten, verwendet man den Befehl POKE -16302,0:

POKE -16302,0 schaltet auf 40 x 48 Punkte (ohne Text)

Um die 4 Zeilen Text am unteren Bildrand wieder zu aktivieren, dient der Befehl

POKE -16301,0 schaltet auf 40 x 40 Punkte + 4 Zeilen Text

Der gesamte Bildschirm wird wieder auf Textausgabe umgeschaltet mit dem Befehl TEXT:

TEXT oder
POKE -16303,0 schaltet von Grafik auf Textausgabe

Um wieder die Grafik einzuschalten, ohne die Grafik zu löschen (wie dies bei GR der Fall ist), verwendet man

POKE -16304,0 schaltet von Text auf Grafikausgabe.

Die niedrigauflösende Grafik belegt den Speicherbereich 1024 bis 2047 (hex $400 bis $7FF, Page 4 bis 7). Je zwei Bildpunkte untereinander (z.B. in Zeile 0 und 1, oder 2 und 3) belegen einen Speicherplatz.

3.14.1.2 80 x 40 (80 x 48) Bildpunkte

Ähnlich wie bei der 80 Zeichen/Zeile Textdarstellung, die aus zwei Darstellungen mit 40 Zeichen pro Zeile entsteht, funktioniert die niedrigauflösende Grafik (80 x 40 oder 80 x 48). Sie aktiviert durch:

PR#3: X= PEEK (-16290): GR schaltet auf 80 x 40 Grafik

Statt der Variablen X kann auch eine andere gewählt werden; wichtig ist nur, daß die angegebene Speicherzelle angesprochen wird.
Analog zur 40 x 40 bzw. 40 x 48 Grafik werden die untersten 4 Zeilen auf Text oder Grafik geschaltet:

POKE -16302,0 schaltet auf 80 x 48 Grafik ohne Text
POKE -16301,0 schaltet auf 80 x 40 Grafik mit Text

Alle anderen Grafikbefehle funktionieren wie bei 40 x 40 oder 40 x48 Grafik. Der Wertebereich der Befehlsargumente ändert sich auf 0 bis 79 für die X-Achse (horizontal). Der Wertebereich der Argumente in Y-Richtung bleibt gleich. Eine Zusammenfassung der für niedrigauflösende Grafik relevanten Speicheradressen und der zugehörigen Funktionen ist im Anhang vorhanden.

3.14.2 Hochauflösende Grafik

Die hochauflösende Grafik (280 x 192 Bildpunkte) ist doppelt vorhanden. Der Benutzer kann zwischen zwei Speicherbereichen wählen, in denen die Grafik-

daten abgelegt sind. Die Inhalte der beiden „Seiten" (Seite 1: 8192 bis 16383, Seite 2: 16384 bis 24575) können auch abwechselnd verwendet werden:

HGR schaltet die 280 x 160 (192) Grafik ein (Seite 1).
HGR2 schaltet die zweite Seite der 280 x 192 Grafik ein.

Beide Befehle löschen den (eventuell vorhandenen) Inhalt der Grafikseiten. Jede der beiden Grafikseiten kann wahlweise 280 x 192 Punkte (Vollgrafik) oder nur 280 x 160 Punkte darstellen. Bei Verwendung von Seite 1 (HGR) sind dann am unteren Bildschirmrand vier Zeilen Text (mit VTAB 20: PRINT ...) möglich, bei Seite 2 (HGR2) dagegen nur über POKE Anweisungen.

Bei Verwendung des HGR Befehls werden die untersten 4 Zeilen des Bildschirms für Text reserviert und die Grafik in 280 x 160 Punkten dargestellt. Mit dem Befehl POKE -16302,0 kann der ganze Bildschirm für Grafik aktiviert werden; mit POKE -16301,0 werden wieder die 4 untersten Zeilen für Textausgabe geschaltet.

POKE -16302,0 schaltet auf 280 x 192 Punkte ohne Text.

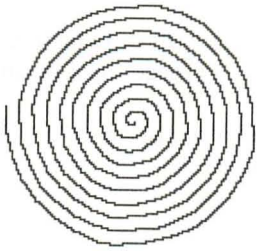
POKE -16301,0 schaltet auf 280 x 160 Punkte (mit 4 Zeilen Text bei Seite 1)

Der Befehl TEXT schaltet aus jeder Grafik wieder auf Textausgabe.

TEXT schaltet auf Textausgabe (24 Zeilen)

Zunächst soll ein kleines Beispiel die Handhabung der Grafik erläutern: Das folgende Programm schaltet die hochauflösende Grafik ein (280 x 160), setzt mit dem Befehl HCOLOR die Farbe und erstellt eine Zeichnung:

```
10  HGR
20  HCOLOR= 7
30  HPLOT 100,100
40  FOR I = 1 TO 60 STEP .2
50  HPLOT  TO 100 + I *  COS (I),100 + I *  SIN (I)
60  NEXT I
70  GET X$: TEXT
```



Die Farbe **HCOLOR=** 7 ist weiß. Insgesamt kann in hochauflösender Grafik mit sechs Farben gezeichnet werden. Bei der Verwendung eines einfarbigen Bildschirms erscheinen die Farben als verschiedene Muster in der Grafik.

Farbe	Nummer
schwarz	0 , 4
grün	1
violett	2
weiß	3 , 7
orange	5
aquamarinblau	6

Wie auch bei niedrigauflösender Grafik, so muß bei hochauflösender Grafik zunächst die Farbe definiert werden (mit HCOLOR= Nummer), sonst wird schwarz gezeichnet (Hintergrundfarbe).
Das obige Beispielprogramm verwendet den Befehl HPLOT. HPLOT ist der wichtigste Befehl für hochauflösende Grafik. Er kann in 4 verschiedenen Varianten verwendet werden:

HPLOT Varianten

HPLOT 12,12	zeichnet Punkt bei 12,12
HPLOT TO 20,49	zeichnet Linie vom Ort des letzten HPLOT's bis 20,49
HPLOT 40,50 TO 122,99	zeichnet Linie von 40,50 bis 122,99

HPLOT 40,50 TO 122,99 TO 7,140 TO 200,33 (etc.)

zeichnet Linienzug mit den angegebenen Eckpunkten.

Der zulässige Bereich der Parameter ist 0 bis 279 in horizontaler Richtung und 0 bis 191 in vertikaler Richtung.

3.14.2.1 Umschalten der Grafikseiten

Die Befehle HGR und HGR2 löschen den Inhalt der jeweiligen Grafikseite. Mit den folgenden POKE Befehlen läßt sich das umgehen:

POKE -16304,0	schaltet Grafik ein
POKE -16297,0	schaltet auf hohe Auflösung
POKE -16300,0	schaltet auf Grafikseite 1

Damit wird die Grafikseite 1 dargestellt. Für Seite 2 sieht das so aus:

POKE -16304,0	schaltet Grafik ein
POKE -16297,0	schaltet auf hohe Auflösung
POKE -16299,0	schaltet auf Grafikseite 2

Um von Seite 1 auf Seite 2 umzuschalten (bzw. umgekehrt), ist nur der jeweils letzte Befehl nötig, also POKE -16299,0 für HGR --> HGR2, bzw. POKE -16300,0 für HGR2 --> HGR.
Anmerkung: Mit diesen beiden Befehlen sind „bewegte Bilder" möglich, ein Beispielprogramm ist unten dargestellt.

3.14.2.2 Linienfolgen (Shapes)

Der Apple //c erlaubt die Definition von Linienfolgen. Linienfolgen sind binär definierte Figuren . Linienfolgen sind wesentlich aufwendiger zu generieren als reine Zeichnungen mit HPLOT Befehlen. Durch den frei wählbaren Vergrößerungsfaktor und Drehwinkel der Darstellung der Linienfolgen sind die zeichnerischen Möglichkeiten aber ungleich größer als bei HPLOTs.

Linienfolgen setzen sich aus acht verschiedenen Grundtypen von Linien zusammen: Je zwei Linien (eine unsichtbare und eine sichtbare) nach oben, nach unten, nach links oder nach rechts.

Um diese acht Grundtypen zu unterscheiden (bzw. um entsprechende Zahlen zwischen 0 und 7 zu speichern), werden 3 Bit benötigt (2 hoch 3 = 8).

Linientyp	Nummer	Binärer Wert
unsichtbar aufwärts	0	000
unsichtbar rechts	1	001
unsichtbar abwärts	2	010
unsichtbar links	3	011
sichtbar aufwärts	4	100
sichtbar rechts	5	101
sichtbar abwärts	6	110
sichtbar links	7	111

In einem Byte (8 Bit) lassen sich 2 Linien (durch je 3 Bit) speichern; die verbleibenden 2 Bit des Bytes können teilweise noch genutzt werden, der Einfachheit halber wird das hier nicht beschrieben.

Das nachfolgende Beispiel erscheint zunächst etwas langwierig, verglichen mit dem dabei erzielten Resultat. Es dient lediglich zur Darstellung der Vorgehensweise. Deshalb wurde bewußt eine einfache Figur gewählt. Die Generierung komplexer Shapes sei dem Anwender überlassen.

Beispiel:
Mit Hilfe der Linienfolgen soll ein Quadrat gezeichnet werden. Ein Quadrat besteht aus einer Linie abwärts, einer nach rechts, einer nach oben und einer nach links, alle sichtbar.

Linienfolge Quadrat : 6, 5, 4, 7 oder binär: 110, 101, 100, 111

Diese drei Linien lassen sich in 2 Byte speichern:

1. Byte: dezimal 6 + 5 * 8 = 46 hex: 00 101 110
5 6

2. Byte: dezimal 4 + 7 * 8 = 60 hex: 00 111 100
7 4

(Die Multiplikation des 2. Linienwertes (5) mit der Zahl 8 bewirkt die Speicherung der drei Bits der Zahl (5) an der richtigen Stelle im 1. Byte, nämlich Bit 4 bis 6).
Diese beiden Bytes (46 und 60) enthalten nun den Code für das Quadrat. Um das Quadrat in der Grafik darstellen zu können, werden diese beiden Bytes ins Memory gepoked, üblicherweise ab Adresse 24576 (hex $6000). Vor dem Shape (so heißt eine definierte Linienfolge) wird ein Vorspann abgelegt, der nachfolgend erklärt wird:

POKE 24576,1	Vorspann
POKE 24577,0	
POKE 24578,4	
POKE 24579,0	
POKE 24580,46	Shape 1
POKE 24581,60	
POKE 24582,0	bezeichnet das Ende von Shape 1

In der Adresse 232, 233 wird noch angegeben, wo die Linienfolge abgelegt wurde (hier ab 24576, hex ab $6000):

POKE 232,0	Angabe der Speicheradresse, ab der
POKE 233,96	das Shape (mit Vorspann) liegt.

Mit den Basic Befehlen **SCALE** und **ROT** wird die Größe und der Drehwinkel angegeben, mit dem Befehl **DRAW** die Zeichnung ausgeführt. DRAW 1 AT... bedeutet „Zeichne Shape 1 an der Stelle ...". (Da das Quadrat das erste von uns definierte Shape ist, wird es von DRAW mit 1 bezeichnet):

```
10 HGR:HCOLOR=7                Farbe weiß
20 SCALE = 20                  Vergrößerungsfaktor 20
30 ROT = 16                    Rotation um 16x5,625 = 90 Grad

40 DRAW 1 AT 100,40            Shape 1 ab Stelle 100,40
                               zeichnen
```

Mit anderern ROT- und SCALE-Werten lassen sich verschieden große Quadrat-Shapes in verschiedenen Richtungen zeichnen.

Der Vorspann des Shapes sei am obigen Beispiel erklärt. Er gibt die Anzahl der Shapes an (hier nur 1) und die Startadresse des Shapes relativ zum Anfang des Vorspanns

POKE 24576,1	Anzahl Shapes
POKE 24577,0	
POKE 24578,4	Abstand 1.Byte 1.Shape zu 24576 (Vorspann)
POKE 24579,0	

Wird ein weiteres Shape angehängt, zum Beispiel 44,44,44,44, so muß der Vorspann wie folgt geändert werden:

POKE 24576,2	Anzahl Shapes
POKE 24577,0	
POKE 24578,6	Abstand 1.Byte 1.Shape zu 24576 (Vorspann)
POKE 24579,0	
POKE 24580,9	Abstand 1.Byte 2.Shape zu 24576 (Vorspann)
POKE 24581,0	

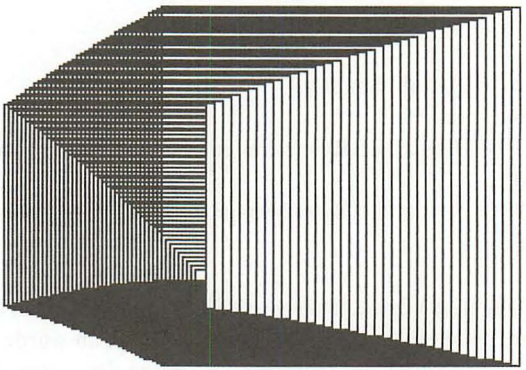
Da durch den längeren Vorspann das erste Shape überschrieben wurde, muß es neu an anderer Stelle eingegeben werden (der Vorspann wurde entsprechend angepaßt). Der Vorspann kann schon bei Definition des ersten Shapes bewußt sehr lang gewählt werden (ungebrauchte Bytes mit Nullen füllen). Das verbraucht zwar Speicherplatz, andererseits werden beim Anhängen neuer Shapes nicht alte Shapes durch den verlängerten Vorspann überschrieben.
Sowohl für die Anzahl der Shapes als auch für die Angabe des Abstandes Vorspann bis Shape sind je zwei Byte im Vorspann reserviert. Darin können Zahlen bis 65535 abgelegt werden (z.B. 260 = 4 + 256 = $04 (Low Byte) + $01 (High Byte)).

Neue Shapes:

POKE 24582,46	1.Byte Shape 1
POKE 24583,60	2.Byte Shape 1
POKE 24584,0	Endezeichen für Shape 1
POKE 24585,44	1.Byte Shape 2
POKE 24586,44	2.Byte
POKE 24587,44	3.Byte
POKE 24588,44	4.Byte
POKE 24589,0	Endezeichen für Shape 2

Das zweite Shape wird analog zum ersten mit DRAW 2 AT X,Y aufgerufen, nachdem zuvor ROT und SCALE definiert wurden (Initialisierung mit POKE 232,0 und POKE 233,96 nicht vergessen !)

```
10 HGR: POKE -16302,0 : REM untere 4 Zeilen Text zu Grafik
20 HCOLOR = 7
40 FOR I = 5 TO 170 STEP 2
50 SCALE = I
60 DRAW 1 AT ABS (I - 95),(I ↑ 2) / 700 - I + 150
70 NEXT I
```



Der Befehl **XDRAW** ist ein Löschbefehl. Er arbeitet analog zum DRAW-Befehl und löscht einen gezeichneten Shape. Da er nicht bei allen Shapes funktioniert (z.B. bei Shapes, die mehrmals an der gleichen Stelle zeichnen) wird er besser durch einen normalen DRAW Befehl ersetzt, nachdem die Farbe HCOLOR= auf 0 (schwarz) bzw. auf die Hintergrundfarbe gesetzt wurde.

Shapes auf Disketten speichern (und einlesen)

Definierte Shapes werden mit
BSAVE Filename, Axxxxx,Lyyyyy
auf Diskette gespeichert (xxxxx = Anfangsadresse und yyyyy = Länge). In unserem Beispiel also mit

BSAVE TEST.SHAPE, A24576, L14

Der BLOAD Befehl liest gespeicherte Shapes wieder von der Diskette ein:

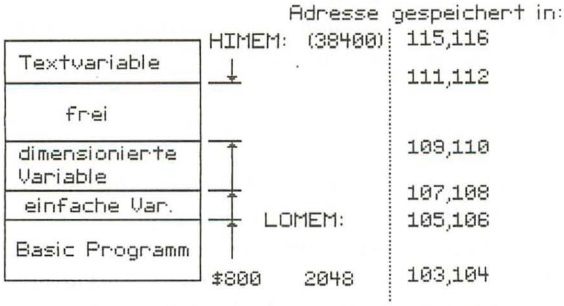
BLOAD TEST.SHAPE

Vor dem Aufruf (mit DRAW ... AT ...) wird in den Speicherzellen 232 (LL) und 233 (HH) die Anfangsadresse der Shapetabelle (hier POKE 232,0: POKE 233,96 als hex Wert ($6000)) gespeichert.

3.14.2.3 Grafik und Speicherplatz

Die Verwendung von Grafik benötigt relativ viel Speicherplatz.
Niedrigauflösende Grafik 40 x 48 belegt den Speicherbereich 1024 bis 2047 (hex 400 bis 7FF). Dieser Bereich dient sonst zur Speicherung des momentanen Textes auf dem Bildschirm.
Hochauflösende Grafik 1 (280 x 192) belegt den Bereich 8192 bis 16383 (hex 2000 bis 3FFF) also 8 kB; HGR2, die zweite Seite der hochauflösenden Grafik belegt 16384 bis 24575 (hex 4000 bis 5FFF).
Die doppelt auflösenden Grafiken 80 x 48 und 560 x 192 belegen die gleichen Speicherbereiche wie die entsprechenden normalauflösenden Grafiken, zusätzlich noch einmal den gleichen Speicherbereich der zweiten 64 kB Bank des Apple //c.

Vor allem bei der Verwendung von HGR und HGR2 sowie bei den 560 x 192 Grafikbereichen kann es zu Komplikationen kommen, da der für die Grafik



Basic Programm und Daten im RAM

Beispiel: HI=PEEK(115)+256*PEEK(116):REM Himem

verwendete Speicherbereich auch von BASIC Programmen benutzt werden kann. Bei der Verwendung von HGR werden zum Beispiel alle Daten im Bereich 8192 bis 16383 gelöscht. Bei anderen Grafiken entprechend.

Normalerweise ist ein BASIC-Programm ab Speicherzelle 2048 aufwärts vorhanden, die dabei verwendeten numerischen Variablen folgen im Anschluß daran. Bis zum Bereich HGR sind also 6 kB frei. Benötigt das Programm inklusive numerischen Daten mehr als 6 kB, so muß HGR2 verwendet werden. HGR2 beginnt ab 16384, für das BASIC-Programm und seine numerischen Daten sind also 14 kB frei.
Mit dem Befehl LOMEM: 16384 bzw. mit LOMEM: 24576 kann der für BASIC-Programme verfügbare Speicher reduziert werden auf den Bereich zwischen LOMEM: und HIMEM: (HIMEM: ist normalerweise 38400). Damit ist der HGR Bereich bzw. auch noch der HGR2 Bereich für Daten von Basic-Programmen unzugänglich.

Die Verwendung des LOMEM: Befehls verschiebt ein schon im Speicher vorhandenes BASIC-Programm nicht! LOMEM: definiert nur den Anfang des Speicherbereichs der Variablen. LOMEM: muß also vor der Verwendung z.B. eines DIM Befehls geschehen, möglichst zu Beginn eines Programms.

3.14.2.4 Bewegte Bilder

Durch ausreichend schnellen Wechsel der Bilder können mit dem Apple zeichentrickfilm-ähnliche Sequenzen erzeugt werden. Der Bildaufbau ist dabei allerdings sehr zeitkritisch, d.h. nur einfache Figuren lassen sich noch mit genügender Geschwindigkeit darstellen.
Das nachfolgend angegebene Beispiel soll den Benutzer zur eigenen Programmierung anregen. Es zeigt einen drehenden Zweitaktmotor. Um die Bildfolge zu beschleunigen, verwendet das Programm folgende 'Tricks':

- der Motor wird schematisch dargestellt, um die Anzahl der zu zeichnenden Bildpunkte zu reduzieren.
- eine Umdrehung des Motors wird in 40 Momentaufnahmen (Bilder) eingeteilt (entspricht einer Drehung um 9 Grad pro Bild).
- alle Bilder werden vor der erstmaligen Darstellung gerechnet.
- dabei werden die Bilder in unbewegte (Gehäuse, Auspuff) und bewegte Teile (Kurbelwelle, Pleuel, Kolben) eingeteilt.
- die errechneten Koordinaten der bewegten Bildpunkte werden pro Bild in Arrays gespeichert.

- die unbewegten Teile des Motors sind in einem weiteren Array abgelegt.
- der Zeichenvorgang erfolgt während der Darstellung des vorhergehenden Bildes auf der jeweils anderen Grafikseite.

Die Berechnung der 40 Bilder dauert einige Sekunden, bevor der Motor zu laufen beginnt.

Beispiel für bewegte Grafik:

```
5  PR# 0: HOME : HTAB 10: VTAB 10: PRINT "Bitte warten"
10  DIM X(43),Y(43),Z(43),A(40),B(40),C(40),D(40)
20 X(0) = 120:Y(0) = 140:PL = 60:H = 25
25 REM

        BERECHNUNG DER 40 EINZELBILDER

30  FOR I = 1 TO 40
35 ALFA = 3.1415 / 19
40 X(I) = X(0) + H *  COS (I * ALFA)
50 Y(I) = Y(0) + H *  SIN (I * ALFA)
60 Z(I) = Y(0) - PL + H *  SIN (I * ALFA)
70 A(I) = X(0) - H / 1.3 *  COS (I * ALFA + .8):B(I) = Y(0) -
    H / 1.3 *  SIN (I * ALFA + .8)
80 C(I) = X(0) - H / 1.3 *  COS (I * ALFA - .8):D(I) = Y(0) -
    H / 1.3 *  SIN (I * ALFA - .8)
100  NEXT I
110  REM

          DARSTELLUNG DER BILDER

190  HGR : HGR2 :I = 1: GOSUB 193
191  POKE 230,32: GOSUB 193: GOTO 200
192  REM

          ZYLINDER

193  HCOLOR= 3: HPLOT X(0) - 22,Y(0) -
 PL + H + 15 TO X(0) - 22,Y(0)
      - PL + H - 10
194  HPLOT X(0) + 22,Y(0) - PL + H - 25 TO X(0) + 120,Y(0) - PL
      + H - 30: HPLOT X(0) + 120,Y(0) -
 PL + H - 10 TO X(0) + 22,Y(0)

      - PL + H - 15 TO X(0) + 22,Y(0) - PL + H + 15
195  HPLOT X(0) - 22,Y(0) - PL + H - 20 TO X(0) - 22,Y(0) - PL
      - H - 15 TO X(0) + 22,Y(0) - PL - H - 15 TO X(0) + 22,Y(0) -
 PL
      + H - 25: HPLOT X(0),Y(0)
```

```
196  HPLOT X(0) - 22,Y(0) - PL + H - 20 TO X(0) - 30,Y(0) - PL
      + H - 15 TO X(0) - 30,Y(0) + H + 5 TO X(0) + 30,Y(0) + H
      + 5 TO X(0) + 30,Y(0) + H + 15 -
 PL TO X(0) + 22,Y(0) + H + 15 - PL
199  RETURN: REM

200  HCOLOR= 0: POKE 230,32
210  GOSUB 999
220  I = I + 2: HCOLOR= 3
230  GOSUB 999
240  POKE  - 16300,0: POKE 230,64:I = I -
 1: HCOLOR= 0: HPLOT X(0)
      - 21,Y(0) - PL - H - 14 TO X(0) + 21,Y(0) - PL - H - 14:
      HPLOT X(0) - 21,Y(0) - PL - H -
 13 TO X(0) + 21,Y(0) - PL - H - 13
245  HPLOT X(0) - 21,Y(0) - PL - H - 12 TO X(0) + 21,Y(0) - PL
      - H - 12: HPLOT X(0) - 21,Y(0) - PL - H - 11 TO X(0) +
      21,Y(0) - PL - H - 11
250  GOSUB 999
260  I = I + 2: HCOLOR= 3
262  IF I <  > 4 THEN 264
263  FOR T = 1 TO 10:A =  PEEK ( - 16336): NEXT T
264  IF I <  > 28 THEN 270
265  HPLOT X(0) - 21,Y(0) - PL - H - 14 TO X(0) + 21,Y(0) -
      PL - H - 14: HPLOT X(0) - 21,Y(0) - PL - H - 13 TO X(0)
      + 21,Y(0) - PL - H - 13
267  HPLOT X(0) - 21,Y(0) - PL - H - 12 TO X(0) + 21,Y(0) -
      PL - H - 12: HPLOT X(0) - 21,Y(0) - PL - H - 11 TO X(0)
      + 21,Y(0) - PL - H - 11
270  GOSUB 999
280  POKE  - 16299,0:I = I - 1: IF I > 38 THEN I = 1:LD =
      LD + 1: PRINT LD
285  IF I = 27 THEN  CALL  - 198
290  GOTO 200
999  HPLOT X(I),Y(I) TO A(I),B(I) TO C(I),D(I) TO X(I),Y(I)
      TO X(0),Y(I) - PL: HPLOT X(0) - 20,Z(I) + 20 TO X(0) -
      20,Z(I) - 10 TO X(0) + 20,Z(I) - 10 TO X(0) + 20,Z(I) +
      20: RETURN
```

Um das Programm (wenn auch nur geringfügig) zu beschleunigen, können die REM's wegelassen werden. Das Programm kommt bei der Darstellung des Motors in eine endlose Schleife. Es kann mit CONTROL-C oder CONTROL-RESET angehalten werden.

3.14.3 Grafik und Disketten

3.14.3.1 Grafik auf Diskette speichern

Grafiken werden binär auf Disketten gespeichert. Dazu dient der Befehle BSAVE. BSAVE schreibt den Inhalt eines definierten RAM Bereichs auf die

Diskette. Die Argumente des Befehls sind Anfangsadresse des Speicherbereichs und Länge des Bereichs.

Beispiel:

BSAVE Bild1, A8192, L8192	speichert HGR Zeichnung als Bild1 auf Diskette

Die Argumente können auch hexadezimal mit vorangestelltem $ Zeichen angegeben werden:

BSAVE Bild1, A$2000, L$2000	ist identisch mit
BSAVE Bild1, A8192, L8192	speichert HGR Zeichnung als Bild1

Für die verschiedenen Grafiken gelten folgende Werte:

Grafik	Anfangsadresse		Länge	
	dezimal	hex	dezimal	hex
GR	1024	$ 400	1024	$ 400
HGR	8192	$2000	8192	$2000
HGR2	16384	$4000	8192	$2000

Hochauflösende Grafik der ersten Seite (HGR), die so auf Diskette gespeichert wurde, kann ins Programm MOUSEPAINT (durch GET PICTURE) eingelesen werden. Dazu muß in MOUSEPAINT die volle ProDOS-Bezeichnung des Files angegeben werden, also Diskettenname und Filename (mit Directorynamen, falls vorhanden).

Beispiel:

Get Picture named: /Bilddisk/Bild1	liest File ins MOUSEPAINT Programm ein.

Für diese Anwendung ist es häufig nützlich, das Bild vorher zu invertieren (Negativ erstellen), siehe dazu Kapitel 3.15.

3.14.3.2 Grafik von Diskette einlesen

Gespeicherte Grafiken werden mit BLOAD „Filename“ wieder eingelesen.

BLOAD Bild1 liest Grafikfile Bild1 von Diskette ein.

Der entsprechende Grafikbefehl (z.B. HGR) darf nicht anschließend an BLOAD durchgeführt werden, weil damit die Grafik gelöscht wird. Entweder wird er vorher durchgeführt (damit wird auf dem Bildschirm sichtbar, wie die Grafik eingelesen wird), oder der Grafikbefehl wird durch POKEs umschrieben (siehe oben).

Beispiel:

```
10 HGR                              schaltet auf Grafik (löscht)
20 PRINT CHR$(4) ; "BLOAD Bild1"    liest Grafikfile ein
.....
```

oder

```
10 PRINT CHR$(4) ; "BLOAD Bild1"    liest Grafikfile ein
20 POKE -16304,0                    schaltet Grafik ein
30 POKE -16297,0                    schaltet auf hohe Auflösung
40 POKE -16300,0                    schaltet auf Grafikseite 1
```

3.15 Die Programmierung der Maus

Der Apple //c enthält bereits die Maus-Firmware. Das sind ROM Routinen (ab $C400), die dem Benutzer die Programmierung erleichtern. Die Maus kann deshalb aus BASIC direkt angesprochen werden.

3.15.1 Programmierung in BASIC

Um mit der Maus zu arbeiten (vor allem Abfrage der Bewegung und der Taste), wird die Maus zuerst initialisiert:

Aktivierung der Maus:

```
20 PRINT CHR$ (4);"PR#4": PRINT CHR$ (1): PRINT  CHR$ (4);"PR#3"
```

PR#4 selektiert Slot 4 (den Mauseingang); CHR$(1) aktiviert die Maus. PR#3 schaltet die Datenausgabe auf 80 Zeichendarstellung zurück. (Natürlich ist auch "PR#0" möglich).

Das folgende Programm zeigt die von der Maus erhältlichen Informationen auf dem Bildschirm:

```
10  REM  Mausdemo -
- MX und MY sind Maussignale in X- und Y-Richtung.
15  REM  MX und MY zwischen 0 und 1024.
16  REM  Funktioniert nicht in 80 Zeichenmodus. BN wird negativ, w
enn
18  REM  Tastaturtaste gedrückt wurde. Sonst keine Eingabe über
19  REM  die Tastatur möglich
20  PRINT  CHR$ (4);"PR#4": PRINT  CHR$ (1): REM   Maus einschal
ten
30  PRINT  CHR$ (4);"PR#0"
35  PRINT  CHR$ (4);"IN#4"
40  INPUT "";MX,MY,BN
50  HTAB 1: VTAB 1: PRINT "Button=";BN,"MX=";MX,"MY=";MY: GOTO 4
0
RUN
```

MX=45 MY=226 BN=32

Das Programm gibt die Bewegung (besser den zurückgelegten Weg) der Maus in X und Y Richtung an. Zudem wird über BN die Taste der Maus abgefragt. Ist BN (absolut) kleiner als 127, so ist die Taste gedrückt, sonst nicht. BN wird negativ, wenn nach der Initialisierung der Maus eine Taste der Tastatur gedrückt wurde.

Im Programm wurde schon erwähnt, daß die Tastatur im Prinzip unwirksam wird. Die Tastatur kann jedoch wieder aktiviert werden, wenn z.B. BN (die Maustaste) negativ wird.

Maus ausschalten:

```
100 PRINT CHR$ (4);"PR#4" :PRINT CHR$(0)
110 PRINT CHR$(4);"PR#3"
120 POKE -16368,0
```

Mit PR#4 wird der Mauseingang angesprochen und die Maus mit CHR$(0) (CONTROL-@) desaktiviert. Anschließend wird mit PR#3 wieder auf 80 Zeichendarstellung geschaltet.
Mit POKE -16368,0 wird der Strobe gelöscht.

Das oben dargestellte Programm hat den Nachteil, daß die Tastatur nicht aktiv ist. Mit dem folgenden Programm läßt sich das umgehen:

```
10  REM  Mausdemo -
- MX und MY sind Maussignale in X- und Y-Richtung.
15  REM  MX und MY zwischen 0 und 1024.
20  PRINT  CHR$ (4);"PR#4": PRINT  CHR$ (1): REM   Maus einschal
ten
30  PRINT  CHR$ (4);"PR#3"
40 MX =  PEEK (1148) +  PEEK (1404) * 256
42 MY =  PEEK (1276) +  PEEK (1532) * 256
44 BN =  PEEK (49251)
50  HTAB 1: VTAB 1: PRINT "Button=";BN,"MX=";MX,"MY=";MY: GOTO 4
0
```

In den Speicherzellen 1148 und 1404 (hex $47C (LL) und $4FC (HH)) ist die X Koordinate und in 1276 und 1532 (hex $57C (LL) und $5FC (HH)) die Y Koordinate abgelegt. BN (Status der Maustaste) kann in 49251 abgefragt werden. Das Programm stellt wiederum die abgelesenen Werte auf dem Bildschirm dar.

3.15.1.1 Grafikprogramme mit der Maus

Die Abfrage der Mausposition erlaubt ein einfaches Zeichnungsprogramm in BASIC. Die Maus steuert einen Lichtpunkt in hochauflösender Grafik. Solange die Maustaste gedrückt ist, wird die Spur der Mausbewegung gezeichnet:

```
10  REM  Mausdemo -
- MX und MY sind Maussignale in X und Y Richtung.
15  REM  Maussignale dienen zum Zeichnen in Hires Grafik.
16  REM  Maus zeichnet, solange die Maustaste gedrückt wird.
17  REM  Mausabfrage in Basic ist langsam, deshalb werden bei
18  REM  Bewegung der Maus einzelne Punkte der Kurven sichtbar.
20  PRINT  CHR$ (4);"PR#4": PRINT  CHR$ (1): REM Maus einschalten
30  PRINT  CHR$ (4);"PR#3": HGR : HCOLOR= 7
40 MX =  PEEK (1148) +  PEEK (1404) * 256
42 MY =  PEEK (1276) +  PEEK (1532) * 256
44 BN =  PEEK (49251)
50  HPLOT MX,MY: IF BN < 127 THEN X = MX:Y = MY: GOTO 40
60  HCOLOR= 0: HPLOT X,Y:X = MX:Y = MY: HCOLOR= 7: GOTO 40
```

Dieses Programm läßt sich relativ einfach erweitern und beschleunigen. Die Erweiterungen können als Ergänzung zum Programm MOUSEPAINT verwendet werden.

```
10  GOTO 150
20  MX =  PEEK (X1) +  PEEK (X2) * C:MY =  PEEK (Y1) +  PEEK
(Y2) * C
30  HPLOT MX,MY: IF  PEEK (D) < 127 THEN 20
40  XX = 0:YY = 0: RETURN
50  MX =  PEEK (X1) +  PEEK (X2) * C:MY =  PEEK (Y1) +  PEEK
(Y2) * C
60  IF R%(MY) < MY THEN R%(MY) = MX
70  IF L%(MY) > MY OR L%(MY) = 0 THEN L%(MY) = MX
80  HPLOT MX,MY: IF  PEEK (D) > 127 THEN 50
90  XX = 0:YY = 0: RETURN
100  IF  PEEK (D) < 127 THEN  GOSUB 20
110  IF  PEEK (T) > 127 THEN  GOSUB 220
120  HCOLOR= 0: HPLOT MX,MY:X = MX:Y = MY: HCOLOR= 7
130  MX =  PEEK (X1) +  PEEK (X2) * C:MY =  PEEK (Y1) +  PEEK
 (Y2) * C
       : PLOT MX,MY
140  GOTO 100
150  REM
            MAUS EINSCHALTEN

160  PRINT  CHR$ (4);"PR#4": PRINT  CHR$ (1)
170  PRINT  CHR$ (4);"PR#3": HGR : HCOLOR= 7
180  X1 = 1148:X2 = 1404:Y1 = 1276:Y2 = 1532:D = 49251:C = 256

      :T =  - 16384
190  DIM R%(160),L%(160)
200  VTAB 22: PRINT "Mit Maustaste zeichnen. Die zwischen der
      Eingabe von S und Druecken der Maustaste umschriebene
      Fläche wird ausgemalt."
210  GOTO 100
220  REM
            TASTATURSIGNAL

230  S =  PEEK (T) - 128
240  IF S = 83 OR S = 115 THEN  GOSUB 50: GOSUB 260: GOTO 1
00
250  POKE  - 16368,0: GOTO 100
260  REM
            AUSMALEN

270  HCOLOR= 7
280  FOR I = 1 TO 160: IF L%(I) > 0 AND R%(I) > 0 THEN  HPLOT
 L%(I),I
      TO R%(I),I:L%(I) = 0:R%(I) = 0
290  NEXT I: POKE  - 16368,0
300  S = 0: RETURN
```

Das Programm soll nicht das MOUSEPAINT ersetzen. Es zeigt, wie mit relativ einfachen Mitteln Ergänzungen zum Programm MOUSEPAINT programmiert werden können. Als Beispiel dient hier das Ausmalen (oder Schraffieren) einer mit der Maus umschriebenen Fläche.

Beschreibung des Programms:
In der ersten Zeile erfolgt ein Sprung nach 150 zur Initialisierung der Maus. Damit wird erreicht, daß die zeitkritischen Zeichnungsroutinen (Zeile 20 bis 90) am Anfang des Programms stehen, wo Sprünge schneller ausgeführt werden.
Alle Konstanten (insbesondere die Adressen der relevanten Speicherzellen) sind Variablen zugeordnet, das beschleunigt den Programmablauf ebenfalls.
Beim Lesen der Y-Position der Maus (Variable MY) kann der zweite Teil weggelassen werden (z.B. in Zeile 20: MY = PEEK (Y1)), da das Programm nur Y-Koordinaten zwischen 0 und 160 erlaubt (kann auf 0 bis 191 erweitert werden). Dazu genügt nämlich ein Byte (0 bis 255). Die X Position muß hingegen aus zwei Bytes (LL + 256 * HH) gelesen werden, sonst kann der Grafikbereich über 255 (256 bis 279) nicht angesprochen werden.
Der Cursor wird von der Maus über den Bildschirm bewegt. Solange die Maustaste gedrückt bleibt (Prüfung in Zeile 100, PEEK (D) < 127, wobei D = 49251), wird die Spur der Mausbewegung gezeichnet.
Neu im Programm sind außer der Beschleunigung der Zeichenroutinen (20 bis 90) die „Ausmalroutine". In Zeile 110 wird zusätzlich der Zustand der Tastatur abgefragt (IF PEEK(T) > 127, wobei T= -16384). Nach dem Drücken einer Taste der Tastatur steht in der Speicherzelle T der ASCII-Wert der gedrückten Taste plus 128. Anhand dieses Wertes wird in Zeile 240 geprüft, ob die Taste S (oder s) gedrückt wurde. Anschliessend wird in den Zeilen 60 und 70 die linke und rechte Kontur der nachfolgend beschriebenen Fläche in den Variablen L() und R() gespeichert.
Wird die Maustaste wieder betätigt, so erfolgt ein Sprung ins Unterprogramm „AUSMALEN" ab Zeile 260. Dort wird die Fläche zwischen den gespeicherten L() und R() Werten (linker und rechter Rand der Fläche) ausgemalt (HCOLOR= 7) und die L(), R() Werte wieder auf 0 gesetzt.

Trotz Optimierung ist das Programm langsam; insbesondere bei der Prüfung des linken und rechten Randes geht viel Zeit verloren, sodaß die Maus langsam bewegt werden muß, um die Fläche ganz auszumalen.
Das Programm kann mit zusätzlichen Befehlen versehen werden, die im Unterprogramm TASTATURSIGNAL (220 bis 250) abgefragt werden können.
Wird das Programm zu groß, so kann es sich mit dem für die Grafik (HGR) reservierten Speicherbereich (dez 8192 - 16383) überschneiden. In diesem Fall werden durch LOMEM:16384 zu Beginn des Programms die Variablen oberhalb des Grafikbereichs abgelegt.

Mit POKE -16302,0 kann der ganze Bildschirm für Grafik verwendet werden (192 x 280, statt 160 x 280). In diesem Fall müssen die entsprechenden

Parameter, insbesondere L%() und R%() für den Bereich 0 bis 191 erweitert werden. Benutzerinformationen unterhalb der Grafik fallen dann weg.

3.15.1.2 Hochauflösende Grafik auf Diskette speichern

Das folgende Programm liest Grafik(binär)files von der Diskette ein, bzw. speichert Grafik auf Diskette ab. Damit können Zeichnungen, die mit MOUSE-PAINT erstellt wurden, eingelesen und geändert werden (z.B. mit dem oben dargestellten Maus-Zeichnungsprogramm).
Die mit Hilfe dieses Programms auf Diskette abgespeicherten Grafikbinärfiles können ins MOUSEPAINT-Programm übernommen und von dort aus geändert oder auf einen Drucker ausgegeben werden.

Zur Kombination des dargestellten Zeichnungsprogramms mit dem MOUSE-PAINT-Programm wird zweckmäßiger ProDOS als Betriebssystem verwendet.

```
1000  REM
            SAVE Hires Grafik to Disk

1010  D$ =  CHR$ (4): REM   CONTROL-D
1020  REM   SAVE Routine
1030  PRINT "Grafik auf Diskette speichern"
1040  INPUT "Bitte Filenamen angeben (oder RETURN):";F$
1050  IF  LEN (F$) = 0 THEN  RETURN
1060  PRINT D$;"BSAVE";F$;",A$2000,L$1FFF"
1070  RETURN
1100  REM
            Load Hires Grafik from Disk

1110  D$ =  CHR$ (4): REM    CONTROL-D
1120  REM   LOAD Routine
1130  PRINT "Grafik von Diskette einlesen"
1140  INPUT "Bitte Filenamen angeben (oder RETURN):";F$
1150  IF  LEN (F$) = 0 THEN  RETURN
1160  PRINT D$;"BLOAD";F$
1170  RETURN
```

Das Programm wird mit GOSUB 1000 (zum Abspeichern), bzw. mit GOSUB 1100 (zum Einlesen) aufgerufen. Der Ansprung kann beispielsweise im oben dargestellten Zeichnungsprogramm zwischen Zeile 240 und 250 eingefügt werden.

3.15.1.3 Grafiken invertieren

Das oben dargestellte Programm zeichnet weiß auf dunklem Grund, während das MOUSEPAINT Programm dunkel auf hellem Grund zeichnet (wie der Drucker auch). Mit dem folgenden Miniprogramm können hochauflösende Bilder der ersten Grafikseite invertiert werden:

```
2000 REM HGR invertieren
2010 FOR I=8192 TO 16383:POKE I,(255 - PEEK(I)): NEXT
```

Das Programm ist nicht sehr schnell. Es zeigt deshalb gut die Aufteilung des Speicherbereichs auf die einzelnen Grafikzeilen. Mit den Grenzen 16384 und 24575 kann die Programmzeile auf für die Grafikseite 2 (HGR2) verwendet werden.

3.15.2 Firmwareroutinen für die Maus

Der Apple //c enthält im ROM ab $C400 Mausroutinen, die von BASIC verwendet werden, aber auch direkt angesprungen werden können.

ROM Maus Firmwareroutinen

Einsprung-adresse	Beschreibung	
$C43D	SETMOUSE Setzt den Mausstatus entsprechend dem im Akkumulator vorhandenen Wert:	
	Akkuwert	
	$00	Maus ausschalten
	$01	Transparent Modus setzen
	$03	Bewegungsinterrupt setzen
	$05	Tasteninterrupt setzen
	$07	Tasten- oder Bewegungsinterrupt setzen
	$08	Maus auschalten, VBLINT aktiv
	$09	Transparent Modus und VBLINT ein
	$0B	Bewegungsinterrupt und VBLINT ein
	$0D	Tasteninterrupt und VBLINT ein
	$0F	Tasten- oder Bewegungsinterrupt + VBLINT ein

VBLINT = Vertikal Blanking

$C4FC SERVEMOUSE
Interrupt Handler
OUTPUT:
Carry=0 Mausinterrupt
Carry=1 anderer Interrupt
Interruptauslöser in $77C:

Bit	Auslöser
7 = 1	Taste gedrückt
6 = 1	Taste bereits seit letzter Abfrage gedrückt
5 = 1	Bewegung seit letzter Abfrage
3 = 1	VBLINT Interrupt
2 = 1	Tasteninterrupt
1 = 1	Bewegungsinterrupt

$C495 READMOUSE
aktualisiert Mausposition und Taste in:

$47C	X Koordinate LL
$4FC	X Koordinate HH
$57C	Y Koordinate LL
$5FC	Y Koordinate HH
$77C	Maus Status

$C484 CLEARMOUSE
setzt Mauspositionen X,Y auf 0

$C46B POSMOUSE
setzt Mauspositionen auf die folgenden Werte

$47C	X Koordinate LL
$4FC	X Koordinate HH
$57C	Y Koordinate LL
$5FC	Y Koordinate HH

$C4B0 CLAMPMOUSE
setzt Grenzwerte für Mausbewegung auf folgende Werte

Akku = 0:	setzt X Grenzwert
Akku = 1:	setzt Y Grenzwert
$478	Minimum LL

	$4F8	Maximum LL
	$578	Minimum HH
	$5F8	Maximum HH

$C46D HOMEMOUSE
setzt Mausposition auf die definierten Minimumwerte X,Y

$C41C INITMOUSE
setzt Maus auf Standardwerte

Mit Ausnahme von SERVEMOUSE werden alle Routinen nach LDX #$C4 und LDY #$40 angesprungen. Nach Beendigung der Routinen sind die Register undefiniert.

Beispiel-Programm:

```
            * MAUS1
            * setting Maus X direction clamping max.
            *
            org $300
clampmouse  equ $c4b0
clearmouse  equ $c484
            ldx #$c4        ;x wert setzen
            ldy #$40        ;y wert setzen
            lda #$01
            sta $5f8
            lda #$00
            jsr clampmouse ;maximum x setzen
            ldx #$c4
            ldy #$40
            lda #$01
            jsr clearmouse ;besser maus auf 0 setzen
            rts
```

Das Assemblerprogramm springt die Mausroutinen CLAMPMOUSE und CLEARMOUSE an. Mit CLAMPMOUSE wird die Obergrenze des X Bewegungsbereichs der Maus von 1023 auf 397 reduziert; mit CLEARMOUSE werden die aktuellen Koordinaten der Maus auf 0 gesetzt.

3.16 Die strukturierte Programmierung

Befehle: **REM, CHAIN, STORE, RESTORE**

Die strukturierte Programmierung scheint vielen Anfängern zunächst übertriebener Aufwand. Bei kurzen, besonders in BASIC typischen Testprogrammen ist das auch so. Kurze Programme, die nur ein Mal verwendet werden, brauchen nicht strukturiert zu werden.
In allen anderen Fällen kann nicht deutlich genug auf den Sinn der strukturierten Programmierung hingewiesen werden. Wenn Programme einen gewissen Grad an Komplexität erreichen, so gilt immer:

PROGRAMME STRUKTURIEREN UND DOKUMENTIEREN

Das Dokumentieren geschieht mit dem Befehl REM (Remark), hinter dem beliebiger Text zur Programmerklärung steht. Das benötigt zwar Speicherplatz, was aber in vielen Fällen keine Rolle spielt. Der durch die REM-Kommentare verbrauchte Speicherplatz kann im Bedarfsfall durch einfache Befehle zurückgewonnen werden (siehe Programmierhilfen).

Strukturierte Programmierung führt zunächst zu einem etwas höheren Aufwand, der sich aber nach kurzer Zeit bezahlt macht, nicht nur im Hinblick auf die eigentliche Programmierzeit und Programmlänge. Besonders in der Testphase wird die leidige Fehlersuche wesentlich verkürzt.
Strukturierung bedeutet im Wesentlichen die allgemein verwendbare Formulierung von Programmteilen, auch Subroutinen, die wiederholt verwendet werden.

3.16.1 Übergabeparameter

Um Subroutinen allgemein verwenden zu können, müssen sogenannte Übergabeparameter definiert werden. Ein Übergabeparameter ist ein Wert, der vom aufrufenden Programmteil dem aufzurufenden Teil (der Subroutine) übergeben wird oder umgekehrt von der Subroutine zum aufrufenden Programmteil als Ergebnis zurückgegeben wird:

```
10 A = 84.4
20 GOSUB 100 : REM WURZELROUTINE
30 PRINT A, B, C
```

```
.
.
99 END

100 REM Subrout Wurzel, In A, Out B und C (Wurzel & kl. Q.zahl)
110 B = SQR (A)
120 I = INT (B)
130 C = I ↑ 2
140 RETURN
```

In Subroutine 100 wird von der Zahl A die Quadratwurzel B und die nächstkleinere Quadratzahl C von A ermittelt. Beispiel:

```
RUN
84.4            9.1869473        81
```

Die Übergabeparameter sind hier die Zahl A, sowie die Zahlen B und C (Ergebnisse).

3.16.2 Unterprogramme

Programmteile, die als Subroutinen verwendet werden sollten, sind insbesondere:

Eingabeprogramme
Ausgabeprogramme
Routinen für den Filezugriff
Hardcopyroutinen

Für die oben genannten Standardfälle wird im Folgenden je ein Beispiel gegeben. Diese Standardfälle können in Programmen verwendet werden. Die Routinen werden getrennt auf Diskette gespeichert und bei Bedarf an neu formulierte Programme angehängt (Siehe Abschnitt Programmierhilfen: Programm RENUMBER, &HOLD, &MERGE). Dabei muß man darauf achten, daß die von diesen Standardsubroutinen verwendeten Zeilennummern nicht im Hauptprogramm vorkommen.

3.16.2.1 Eingabeprogramm

```
9100  REM    - Eingabeprogramm ; E = Anzahl Fragen
9110  REM    -- Antworten in E$(E) ; Fragen in F$(E)
```

```
9120  REM  Titel in T1$,T2$,T3$
9130  HOME : IF  LEN (T1$) > 0 THEN  PRINT T1$
9140  IF  LEN (T2$) > 0 THEN  PRINT T2$
9150  IF  LEN (T3$) > 0 THEN  PRINT T3$
9160  PRINT : PRINT :V% =  PEEK (37)
9170  FOR I9 = 1 TO E
9180    PRINT  SPC( 30 -  LEN (F$(I9)));F$(I9);":";: INPUT
        "";E$(I9)
9190  NEXT I9
9200  PRINT : INPUT "Sind alle Eingaben richtig?, N=nein, sonst
      ja :";X$
9210  IF X$ = "N" OR X$ = "n" THEN 9230
9220  T1$ = "":T2$ = "":T3$ = "": FOR I9 = 1 TO E:F$(I9) = "":
      NEXT I9:I9 = 0: HOME : RETURN
9230  VTAB  PEEK (37): PRINT  SPC( 60): VTAB V% + 1: REM  -
      Korrektur
9240  FOR I9 = 1 TO E
9250    HTAB 32: INPUT "";X$: IF  LEN (X$) <  > 0 THEN E$(I9)
        = X$
9260    VTAB  PEEK (37): HTAB 32: PRINT E$(I9)
9270  NEXT I9: GOTO 9200
```

Das ist ein Beispiel für ein Standardeingabeprogramm. Es kann separat auf Diskette gespeichert werden und z.B. mit dem DOS 3.3 Programm RENUMBER in einem eigenen Programm eingesetzt und dort mit GOSUB aufgerufen werden. Das Eingabeprogramm verwendet folgende Übergabeparameter:

T1$,T2$,T3$:	Titel
E :	Anzahl Fragen, die beantwortet werden sollen. (Maximal 9)
F$(1) bis F$(E):	Texte zu den Fragen 1 bis E mit maximal 30 Zeichen pro Frage
E$(1) bis E$(E):	Antworten zu den Fragen 1 bis E.

Das Programm löscht den Bildschirm und schreibt die Titel mit zwei Freilinien (Zeile 9100 bis 9160).
Anschließend werden die Fragen gestellt und die Anworten entgegengenommen. Danach erfolgt in Zeile 9200 der Hinweis auf die Korrekturmöglichkeit. Die Korrektur läßt die vorher gestellten Fragen mit ihren bereits gegebenen Anworten auf dem Bildschirm und setzt den Cursor auf den ersten Buchstaben der jeweiligen Antwort. Mit RETURN kann diese akzeptiert werden, andernfalls wird sie überschrieben.
Sind alle Fragen richtig beantwortet, so werden die verwendeten Fragen und weitere Parameter gelöscht und der RETURN zum aufrufenden Programmteil durchgeführt. Dort können die Anworten weiterverarbeitet werden, die in E$(1) bis E$(E) vorhanden sind.

3.16.2.2 Ausgabeprogramm für Textdaten auf den Bildschirm

```
9300  REM - Ausgabe Text auf Bildschirm >>> DIM TA$(22) <<<
9310  REM - TA$(1) - TA$(22) ist Text in Zeilen 1 bis 22
9320  REM - ZZ = Zeichenzahl/Zeile ZZ = 40 oder ZZ = 80;
      TA$ entsprechend maximal 40 oder 80 lang
9330  IF ZZ > 40 THEN  PRINT  CHR$ (4);"PR#3"
9340  IF ZZ < 40 THEN  PRINT  CHR$ (4);"PR#0"
9350  HOME : PRINT : FOR I = 1 TO 22: PRINT TA$(I): NEXT I
9360  FOR I = 1 TO 22:TA$(I) = "": NEXT I
9370  RETURN
```

Das Programm verwendet die Variable ZZ, um die Darstellung mit 40 bzw. 80 Zeichen pro Zeile vorzubereiten. Der Bildschirm wird gelöscht und die Textdaten in den Variablen TA$(1) bis TA$(22) ausgegeben. Die Textvariablen müssen vor der Verwendung definiert und dimensioniert werden. Sie dürfen nicht länger als 80 bzw. 40 Zeichen sein. Vor der Rückkehr ins aufrufende Programm werden die Textvariablen gelöscht.

3.16.2.3 Sequentieller Zugriff auf Diskette

```
9500  REM - Sequentielle Textausgabe auf Disk
9510  REM - F$ = Filename; SZ = Anzahl Strings;
      TA$( ) = String
9520  REM - 9530 weglassen, wenn am Programmanfang
      definiert
9530  D$ =  CHR$ (4):O$ = "OPEN":W$ = "WRITE":
      C$ = "CLOSE":P$ = "PRINT"
9540  PRINT D$;O$;F$: REM  - OPTIONAL
9560  PRINT D$;W$;F$
9570  FOR I = 1 TO SZ: PRINT TA$(I): NEXT I
9580  PRINT D$;C$;F$
9590  SZ = 0: REM  - Strings nicht gelöscht
9595  RETURN

9600  REM - Sequentielles Texteinlesen von Disk
9610  REM - F$ = Filename; SZ = Anzahl Strings;
      TA$( ) = String
9620  REM - 9630 weglassen, wenn am Programmanfang
      definiert
9630  D$ =  CHR$ (4):O$ = "OPEN":R$ = "READ":C$ = "CLOSE"
9640  PRINT D$;O$;F$: REM  OPTIONAL
9650  PRINT D$;R$;F$
9660  FOR I = 1 TO SZ: INPUT TA$(I): NEXT I
9670  PRINT D$;C$;F$
9680  RETURN
```

Das erste Programm (Zeilennummer 9500 und folgende) eröffnet das Textfile F$ (das vorher definiert werden muß) auf der Diskette. Die in TA$( ) vorhandenen Texte werden dann sequentiell auf das File geschrieben. Die Anzahl der Strings (vorher dimensionieren) ist durch SZ (Stringzahl) gegeben.

Entsprechend dem Programm 9500 bis 9595 (Schreiben auf Diskette) wird im Programm 9600 bis 9680 vom File F$ gelesen. Die Anzahl der einzulesenden Strings muß in SZ angegeben werden. Die Strings werden in die Variablen TA$() geschrieben (vorher dimensionieren).

3.16.2.4 Text-Hardcopy auf Drucker

```
9800 REM HARDCOPY 40 ZEICHEN TEXT AUF DRUCKER, SN= SLOTNR
 DRUCKER
9810 PRINT CHR$(4);"PR#";SN
9820 IF SN = 1 THEN PRINT CHR$(9);„80N„: REM DISABLE ECHO
9830 FOR I = 0 TO 2
9840 FOR J = 0 TO 7
9850 FOR K = 0 TO 39
9860 PRINT  CHR$ ( PEEK (1024 + 128 * J + K + I * 40));: NEXT K
9870 PRINT: NEXT J,I
9880 PRINT CHR$(4);"PR#0"
9890 RETURN
```

Dieses Unterprogramm erzeugt eine Kopie des Bildschirms mit 40 Zeichen pro Zeile auf dem Drucker. SN enthält dabei die Nummer des seriellen Anschlusses des Druckers (SN = 1 oder SN = 2).

Grundsätzlich werden strukturierte Programme kürzer als unstrukturierte, weil praktisch keine Teile mehrfach vorkommen. In vielen Fällen führt die Strukturierung auch zu einer schnelleren Abarbeitung.
Programme lassen sich durch geeignete Techniken besonders kompakt oder besonders schnell machen. In der Regel widersprechen sich aber diese beiden wichtigen Forderungen an Programme.

3.16.3 Programmverkettung, Chaining

Diese Möglichkeit der Programmierung ist nur unter ProDOS gegeben oder unter DOS 3.3 mit Integer-Basic. Die Ausführungen beschränken sich auf das Vorgehen unter ProDOS.

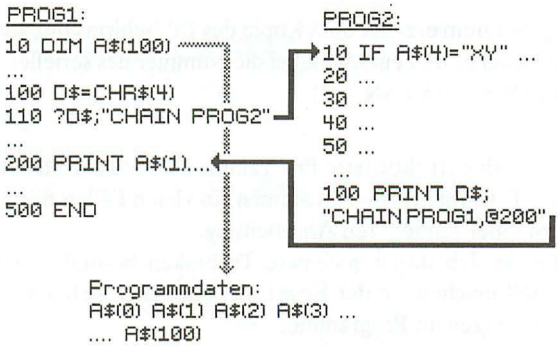
Drei neue Befehle des Betriebssystem ProDOS ermöglichen das Chaining. Chaining heißt Verkettung. Dabei werden Programme oder Programmteile verkettet oder aneinandergehängt, anstatt ein einziges großer Programm zu verwenden. Die drei ProDOS Befehle sind:

CHAIN Filename — liest neues Programm von der Diskette und startet es. Die bestehenden Daten des alten Programms werden nicht gelöscht.

STORE Filename — speichert alle Daten (bisher verwendete Variable) eines Programmes auf Diskette.

RESTORE Filename — liest die mit STORE auf Diskette gespeicherten Programmdaten wieder ein.

Ein kurzes Beispiel soll die Vorgehensweise verdeutlichen:
Das erste Programm wird unter dem Namen „HAUPTPROGRAMM“ auf Diskette gespeichert; das zweite Programm unter dem Namen „PROG2“. Das erste Programm ruft mit CHAIN das zweite Programm PROG2 auf, dieses wiederum ruft das erste Programm auf.



"CHAINING" zweier Programme, gemeinsame Verwendung der Daten.

HAUPTPROGRAMM:

```
10  REM  Chaining demo
20  REM  Hauptprogramm generiert Daten, speichert diese als File
30  REM  Prog.var ab, und ruft das Programm PROG2 auf.
40  REM  PROG2 verarbeitet die Daten, und liest die
50  REM  Originaldaten wieder ein. Anschließend wird das
60  REM  Hauptprogramm aufgerufen und fährt fort.
70  IF A > 0 THEN 100: REM  - chaining
80  DIM X(10):A = 1: HOME
90  FOR I = 1 TO 10:X(I) = 2 * I: NEXT
100  PRINT : PRINT "Hallo, ich bin das Hauptprogramm"
110  PRINT : PRINT "Meine Daten sind:": PRINT
120  FOR I = 1 TO 10: PRINT X(I);" ";: NEXT
130  IF A > 1 THEN  END
140  PRINT  CHR$ (4);"STORE Prog.var"
150  PRINT  CHR$ (4);"CHAIN PROG2"
```

PROG2:

```
10  REM  chaining demo
20  REM  PROG2 verarbeitet Daten vom Hauptprogramm
30  REM  liest die alten Daten wieder ein
40  REM  und ruft das Hauptprogramm wieder auf
50  PRINT : PRINT : FOR I = 1 TO 70: PRINT "*";: NEXT :
    PRINT : PRINT "Hallo, ich bin PROG2"
60  PRINT : PRINT "Die alten Daten waren:": PRINT
70  FOR I = 1 TO 10: PRINT X(I);" ";: NEXT
80  PRINT : PRINT : PRINT "Die neuen Daten sind:": PRINT
90  FOR I = 1 TO 10:X(I) = 2 * X(I): NEXT : FOR I = 1 TO
    10: PRINT X(I);" ";: NEXT
100  PRINT  CHR$(4);"RESTORE Prog.var"
110 A = 2: PRINT : FOR I = 1 TO 70: PRINT "*";: NEXT :
    PRINT
120  PRINT  CHR$ (4);"CHAIN HAUPTPROGRAMM"
```

Die von PROG2 veränderten Daten können natürlich auch ins Hauptprogramm übernommen werden. Dazu wird Zeile 100 einfach weggelassen.

Der STORE Befehl benötigt bei großen Datenmengen etwas Zeit, bevor sie auf Diskette geschrieben werden, weil die Programmdaten komprimiert werden.

In Applesoft dürfen Arrays nur einmal dimensioniert werden. Wird in einem Programm z.B. die Anweisung DIM A$(20) ein zweites Mal verwendet, so kommt es zum „REDIM'D ARRAY ERROR". Beim Chaining kann dies verhindert werden, indem die Dimensionierungen in einem Programmteil zusammengefaßt werden und dieser Programmteil nur einmal aufgerufen wird. Daneben kann der Fehler über eine Fehlerabfangung (ONERR) eliminiert werden.

Eine andere Möglichkeit, beim Chaining die Mehrfachdimensionierung zu vermeiden, ist folgende:

RUN PROG1

wobei PROG1 die Zeile 10 DIM A$(.... enthält. PROG1 führt (in Zeile 10) die Dimensionierung aus. PROG1 ruft andere Programmteile mit CHAIN auf, wird aber selbst nur wieder mit dem Befehl

PRINT CHR$(4) ; "CHAIN PROG1," ; CHR$(64) ; "20"

aufgerufen. Damit wird PROG1 ab der angegebenen Zeile (hier 20) ausgeführt, die Dimensionierung in Zeile 10 findet nicht mehr statt. CHR$(64) kann bei Verwendung der US-Tastatur durch Shift-2 generiert werden.
Chaining wird besonders bei Verwendung der RAM-Disk wegen der hohen Zugriffsgeschwindigkeit interessant. Außerdem muß in diesem Fall nur einmal auf die Programmdiskette zugegriffen werden. Die Vorgehensweise ist in Abschnitt 3.22 beschrieben.

3.17 Symbolische Sprungadressen – strukturiertes Basic

APPLESOFT erlaubt keine symbolischen Sprungadressen für GOTOs und GOSUBs. Mit dem folgenden Programm können auch Namen als Sprungadressen verwendet werden. Das Programm wurde aus Heft 12,84 der Zeitschrift mc entnommen und zur leichteren Handhabung etwas geändert.
Das Maschinen-Programm verwendet den Basic-Befehl &. Dieser & Befehl ist für Erweiterungen von APPLESOFT Basic vorgesehen. Normalerweise hat der & Befehl keine Funktion. Durch Änderung einer Sprungadresse in den Speicherzellen 1014 und 1015 kann mit dem & Befehl ein Benutzerprogramm gestartet werden. Das geschieht hier automatisch.
Das Programm wird im Monitor eingegeben; der Monitor wird angesprungen mit:

CALL -151

und meldet sich mit dem Stern *.

Ab Speicherzelle 300 werden die folgenden Zahlen gespeichert:

```
*300:A9 0B 8D F6 03 A9 03 8D
*308:F7 03 60 A5 67 85 FA A5
*310:68 85 FB 20 B7 00 C9 B0
*318:D0 07 A9 00 85 FC 18 90
*320:0B A2 FF 85 FC C9 AB F0
*328:03 4C C9 DE 20 B1 00 C9
*330:22 D0 F6 A5 B8 85 FD A5
*338:B9 85 FE 18 A5 FD E9 04
*340:85 FD B0 02 C6 FE A0 05
*348:B1 FA C9 3E F0 15 A0 00
*350:B1 FA AA C8 B1 FA 86 FA
*358:85 FB A0 01 B1 FA D0 E6
*360:4C 7C D9 C8 B1 FD D1 FA
*368:F0 F9 C9 22 D0 E0 A0 02
*370:B1 FA 85 50 C0 B1 FA 85
*378:51 A5 FC F0 03 4C 41 D9
*380:A9 03 20 D6 D3 A5 B9 48
*388:A5 B8 48 A5 76 48 A5 75
*390:48 A9 B0 48 20 41 D9 4C
*398:D2 D7
```

Mit CONTROL-C oder 3D0G erreicht man vom Monitor aus wieder den Basic Interpreter und speichert das Maschinenprogramm mit

BSAVE SYMBASIC, A768, L154

(oder einem anderen Namen) ab.

Um das Programm zu aktivieren (&-Vektor setzen) wird es mit

BRUN SYMBASIC

gestartet. Danach kann ein Basic Programm geschrieben oder eingelesen bzw. gestartet werden, welches die neuen &GOTO sowie &GOSUB Befehle enthält. Ein Testprogramm zeigt die Funktionsweise des erweiterten Basic:

```
10 &GOSUB "Unterprogramm"
20 PRINT "Hier ist das Hauptprogramm"
30 END
100 REM>Unterprogramm
110 PRINT "Hallo, hier ist das Unterprogramm"
120 GET X$: RETURN
RUN
```

Hallo, hier ist das Unterprogramm
Hier ist das Hauptprogramm

```
10 D$=CHR$(4): REM CONTROL-D
20 ?D$;"BRUN SYMBASIC"
...
100 &GOSUB "ALARM"
110  FOR J=J1 TO ...
...
...
1000 REM>ALARM
1010 FOR I=1 TO 5:? CHR$(7):NEXT
1020 RETURN
```

Strukturiertes Programmieren mit Text als Zielbezeichnung bei Sprungbefehlen.

Das & Zeichen bewirkt einen Sprung in das Maschinenprogramm SYMBASIC, wo geprüft wird, ob ein GOSUB oder ein GOTO folgt. Ist das der Fall, so wird im Programmtext das betreffende Unterprogramm bzw. das Sprungziel mit Hilfe des Namens gesucht. Aus Zeitgründen werden nur Zeilen geprüft, die als zweites Zeichen ein „größer"-Zeichen (>) enthalten. Deswegen muß in der Zielzeile (hier 100) das „größer"-Zeichen direkt hinter dem REM eingeben werden, ohne Leerzeichen dazwischen. Das REM in der Zielzeile ist notwendig, um die Interpretation der Zeile zu verhindern, die einen SYNTAX ERROR zur Folge hätte.

Beispiel einer Zielzeile:

100 REM > Unterprogramm

Im Listing wird vom Interpreter ein Leerzeichen eingefügt (deshalb die Zeile nicht wieder mit dem Escape Editor über das Listing einlesen!).
Ist nach dem Namensvergleich das Ziel gefunden, so erfolgt der Sprung. Statt &GOSUB kann auch &GOTO verwendet werden. Insgesamt sind folgende Sprünge möglich:

Mögliche Sprünge mit Symbasic:

&GOSUB "Zielbezeichnung"
IF ... &GOSUB "Zielbezeichnung"

&GOTO "Zielbezeichnung"
IF ... &GOTO "Zielbezeichnung"

Die Befehlskombinationen ON ... GOSUB und ON ... GOTO arbeiten **nicht** mit dem & Befehl.

Unterprogramme wie die oben angegebenen Eingabe- oder Diskroutinen können mit Hilfe der &GOSUB Erweiterung durch entsprechende REM Zeilen am Anfang der Unterprogramme ohne Mühe in eigene Programme eingebunden werden.
Die &GOSUBs können wie normale GOSUBs bis zu 24 mal ineinander geschachtelt werden.

Das Beispielprogramm zeigt die Vorgehensweise bei der Verwendung der Syntax:

```
10   REM  - Demoprogramm für symbolische Sprungadressen
20   REM  - Spruenge mit &GOSUB "Ziell"
30   REM  - Zieladresse mit Zeilennr. REM> Ziell
40   REM  - funktioniert auch mit &GOTO
50   &  GOSUB "Titelprogramm"
60   &  GOSUB "INFO"
70   GET X$
80   &  GOSUB "Hurra"
90   END
1000   REM >Titelprogramm
1010   PRINT  CHR$ (4);"PR#3"
1020   PRINT : PRINT "Demoprogramm fuer STRUKTURIERTES
       BASIC"
1030   RETURN
1500   REM >INFO
1510   VTAB 5: HTAB 5
1520   PRINT "Das Programm fuehrt nun einen Sprung aus."
1530   HTAB 5: PRINT "Die Absprungzeile heißt:": PRINT
1540   HTAB 5: INVERSE : LIST 80: NORMAL
1560   PRINT : HTAB 5: PRINT "Bitte RETURN druecken, um den
       Sprung auszuloesen:";
1570   RETURN
2000   REM >Hurra
2010   PRINT : HTAB 5: PRINT "Die Zielzeile heißt:"
2020   PRINT : HTAB 5: INVERSE : LIST 2000: NORMAL
2030   PRINT : HTAB 5: PRINT "Ich bin angekommen"
2040   PRINT : HTAB 5: PRINT "Das Freizeichen in Zeile 2000
       nach REM wird vom LISTING produziert."
2050   PRINT : HTAB 5: PRINT "Die Eingabe der Zeile
       lautet:";: INVERSE : PRINT "2000 REM>Hurra
       <RETURN>";: NORMAL
2060   RETURN
```

Das Programm ist selbsterklärend. Es zeigt den Sprung an und stellt die relevanten Zeilen dar.

3.18 Kompakte Programme

Große Datenmengen führen häufig zu Komplikationen mit dem Speicherbereich des Apple //c. Wie beschrieben, sind nur 64 kB des Speichers sofort adressierbar, die zweite 64 kB Bank kann nur über zusätzliche Routinen erreicht werden. Die ersten 64 kB sind zudem teilweise durch Systemsoftware belegt, sodaß der Benutzer nur über etwa 36 kB freien Speicher verfügen kann, bevor er in das System eingreifen muß.
Eine Methode, Programme zu verkürzen, um Speicherplatz zu gewinnen, ist die beschriebene Strukturierung. Weiter kann Speicherplatz gespart werden, indem man die Zahl der verwendeten Variablen reduziert.
Die in FOR NEXT Schleifen verwendeten Zählvariablen können mehrfach verwendet werden (nicht bei ineinander geschachtelten Schleifen). Allerdings wird dadurch das Programm unübersichtlicher. Werden diese Zählvariablen immer mit I1,I2 etc. bezeichnet (I1 für die jeweils äußerste Schleife, I2 für die nächst innere usw.), so wird Speicherplatz gespart und gleichzeitig die Lesbarkeit der Programme verbessert, da man aus dem Namen der Variablen (z.B. I2) auf deren Sinn schließen kann.
Ein großer Speicherplatzverbraucher ist Text. Im Programm notwendige Erklärungen und Hinweise sollten nur einmal definiert werden z.B.:

T1$ = "Bitte RETURN drücken"
T2$ = "Unzulässige Division durch 0"

Der Text kann an jeder Stelle des Programms verwendet werden, benötigt aber nur einmal den Speicherplatz (7 Byte + Anzahl der Zeichen).
Dimensionierte Integerzahlen benötigen 2 Byte Speicherplatz, Gleitkommazahlen dagegen 5 Byte. Besonders bei großen Datenmengen (z.B. in Arrays) sollten deshalb wenn möglich Integerzahlen verwendet werden.
Auch Arrays, deren Ausdehnung kleiner als 11 ist, sollten dimensioniert werden, da sie sonst automatisch auf den Wert 11 (entprechend z.B. einem DIM A$(10)) dimenioniert werden, und damit größer als nötig sind.
Zeilennummern benötigen 5 Byte; es können mehrere Befehle in eine Zeile geschrieben werden, um die Zeilenzahl zu reduzieren. Dies sollte auf sinngemäß zusammengehörige Befehle beschränkt werden, um die Übersichtlichkeit nicht zu stark zu verschlechtern.

Remarks (REM) benötigen ebenfalls Speicherplatz. Mit speziellen Befehlen von Hilfsprogrammen (z.B. &Compress im Programm APA) können die Texte hinter den REM Befehlen einfach gelöscht werden. Ein solches „komprimiertes“ Programm wird nur zur Ausführung benutzt. Für Test und Entwicklung sollte immer eine nicht komprimierte Version (mit REM Kommentaren) gesondert auf Diskette gespeichert bleiben.
Bei der Verwendung von Stringbefehlen werden vom APPLESOFT Interpreter Kopien der Strings angefertigt, die ebenfalls Speicherplatz brauchen. Mit dem Befehl

```
CALL 58400        (Garbage Collection)
```

oder

```
GK = FRE(0)
```

können die nicht mehr benötigten Strings gelöscht werden. Da die Ausführung dieser Befehle unter DOS 3.3 nicht sonderlich schnell ist, sollte CALL 58400 oder der FRE()- Befehl (der ebenfalls die Garbage Collection auslöst) nicht zu häufig durchgeführt werden (nicht in oft gebrauchten Schleifen).
ProDOS enthält den Befehl FRE. Er wird von einem Programm durch

```
PRINT CHR$(4); "FRE"
```

aufgerufen. Der ProDOS Befehl löst ebenfalls eine Garbage Collection aus, deren Routinen aber wesentlich schneller sind, als das FRE(0) des Applesoft Interpreters.

3.19 Schnelle Programme

Die Arbeitsgeschwindigkeit des APPLESOFT-Interpreters ist im Vergleich mit BASIC-Interpretern anderer Computer relativ hoch. Programme in Sprachen wie Pascal oder Forth sind aber deutlich schneller.
Geschwindigkeitsprobleme treten allerdings nur in bestimmten Programmen auf.
Programme, die im Wesentlichen mit dem Benutzer kommunizieren (z.B. Textverarbeitungsprogramme), sind meist unkritisch, da der Mensch sehr viel langsamer ist als der Computer.

Bei Programmen, die viele Rechenoperationen durchführen und nur ab und zu wieder mit dem Benutzer Kontakt haben, sollte dagegen besonderer Wert auf Schnelligkeit gelegt werden, um Wartezeiten des Benutzers zu reduzieren. Das hat nicht nur mit Bequemlichkeit zu tun. Mit zunehmender Antwortzeit des Computers (kritische Zeit etwa 1 Sekunde) nimmt die Fehlerhäufigkeit des Benutzers stark zu, weil dieser dann zwischenzeitlich an andere Sachen denkt.

Der Apple //c führt etwa 1 Million Operationen pro Sekunde aus. Ein BASIC-Befehl besteht aus einigen bis einigen hundert dieser Operationen. Daraus folgt, daß die Ausführungszeit eines Programms ungefähr mit der Zahl der durchgeführten Befehle steigt.
Die Geschwindigkeit eines Programmes kann gesteigert werden, wenn man die folgenden Regeln einhält:
Für numerische Konstanten und Texte, die mehrmals vorkommen, besonders in Schleifen, Variablen verwenden:

```
PI = 3.141529
S$ = "Summe :"
```

(Der Interpreter kann Variablen wesentlich schneller „lesen" als Zeichenketten.)
In oft durchlaufenen Schleifen sollen nur die Befehle ausgeführt werden, die wirklich notwendig sind. Befehle, die im Prinzip nur einmal gebraucht werden, sind dazu aus der Schleife zu entfernen.
Variable, die besonders häufig gebraucht werden, sollten als erste definiert werden. Die Speicherung der Variablen (in einem eigenen Speicherbereich) geschieht in der Reihenfolge ihres erstmaligen Vorkommens im Programm. Der APPLESOFT Interpreter sucht bei Bedarf die Variablenliste von Anfang an durch, und findet natürlich die am Anfang erwähnten Variablen dann schneller als die später definierten.
Häufig verwendete Unterprogramme oder Schleifen (die nicht zur Kommunikation mit dem Benutzer dienen) sollten möglichst am Programmanfang stehen. Ähnlich wie bei den Variablen wird bei der Verwendung eines Sprungbefehls (GOTO, GOSUB, NEXT) vom Programmanfang aus nach der richtigen Zeilennummer gesucht; also werden die am Anfang stehenden Zeilen schneller gefunden als die späteren.
Ein NEXT Befehl wird schneller ausgeführt, wenn er kein Argument hat; die Argumente dienen im Prinzip nur dem Benutzer zur Kennzeichnung.

3.20 BASIC Compiler

Das im Apple //c vorhandene Applesoft Basic ist eine Interpretersprache. Ein Basic-Programm wird während seiner Ausführung übersetzt. Diese Technik erlaubt die direkte Ausführung der Programme nach der Eingabe oder Änderungen des Programms.
Die laufende Übersetzung des Programms wirkt sich nachteilig auf die Arbeitsgeschwindigkeit aus. Das macht sich umso stärker bemerkbar, je länger das Programm ist.
Für Applesoft werden verschiedene Compiler angeboten. Compiler sind Übersetzungsprogramme, die den vom Benutzer eingebenen Basic-Text in Maschinenprogramme übersetzen (compilieren). Die Maschinenprogramme sind mechanisch hergestellte Assemblerprogramme. Deshalb erreichen sie nicht die Geschwindigkeit und Kompaktheit gut programmierter Assemblerprogramme, sie sind jedoch normalen Basic Interpreterprogrammen deutlich überlegen. Zudem bieten sie dem Programmierer die Möglichkeit, seine Programme zu schützen, da die compilierten Programme äußert mühsam zu lesen sind und deshalb nur schwer geändert werden können.

Compilierte Basic Programme sind in der Regel etwa 2 bis 3 mal so schnell wie interpretierte.
Der Geschwindigkeitsvorteil rührt vor allem von Wertzuweisungen (z.B. A = B) und von Sprüngen (z.B. GOTO 100) her, die beide compiliert ca. 10 bis 20 mal so schnell ausgeführt werden wie interpretiert.

Da die Compiler auf „serienmäßiges" Applesoft Basic eingestellt sind, können eigene Basicerweiterungen (z.B. mit & Befehlen) zu Schwierigkeiten beim Compilieren führen.
Die angebotenen Compiler sind im Allgemeinen für DOS 3.3 ausgelegt. Beim Kenntnisstand Sommer 1985 gab es noch keinen unter ProDOS funktionsfähigen Applesoft-Compiler.
Compilierte Basic-Programme sind in der Regel größer als der originale Basic-Text. Oft wird zur Ausführung der compilierten Programme ein Maschinencode benötigt, der zusätzlich einige kB belegt.

3.21 Programmierhilfen

Befehle: **TRACE, NOTRACE, STOP, CONT, MON, NOMON, SPEED=, CONTROL-C, CONTROL-S**

Bevor die in der Software vorhandenen Programmierhilfen angesprochen werden, soll ein kurzer Vorgehensplan für die Entwicklung größerer Programme gegeben werden:

3.21.1 Vorgehensplan für die Programmentwicklung

1. Definition der Aufgabe.
In diesem ersten Schritt wird erarbeitet, was das Programm machen soll, aus welchen Daten welche Ergebnisse erzeugt werden sollen; auf Papier wird der grobe Ablaufplan des Programms festgelegt.
2. Algorithmen suchen.
Anschließend wird ermittelt, mit welchen Rechnungen diese Ergebnisse erzielt werden können, in welcher Form Eingabedaten, Zwischenergebnisse oder Resultate gespeichert werden sollen. In manchen Fällen muß festgestellt werden, am besten durch Versuche mit kurzen Testprogrammen, ob gewisse Routinen zeit- oder speicherplatzkritisch sind, bzw. wie diese Probleme gelöst werden können.
3. Aufteilung des Programms
Abschätzung, welche Teilprogramme und Subroutinen gebraucht werden; wieviel Programmzeilen für jedes davon notwendig sind. Aufteilung der Zeilennummern (z.B Abfrage Input Zeile 4000 bis 4399, Daten lesen von File Zeile 8000 bis 8199, etc).
4. Formulierung der wichtigen Schritte
Schriftliche Formulierung vor allem der bisher noch nicht verwendeten wichtigen Berechnungs- oder Verarbeitungsschritte in APPLESOFT.
5. Programm schreiben
Auf Papier werden die einzelnen Zeilen von Programmteilen ausformuliert, gleichzeitig mit Kommentaren (REM) versehen.
6. Programm eingeben
Programm teilweise oder ganz eingeben, listen, und auf SYNTAX (Rechtschreibungs-) Fehler prüfen (eventuell mit Textverarbeitung).
7. Programm testen
Zunächst Programmteile, später das Gesamtprogramm auf logische, Syntax- und andere Fehler überprüfen.

In der gesamten Programmentwicklung wird vom Groben zum Detail gearbeitet (Top-Down-Approach). Die Dokumentation des Programms umfaßt die bei den einzelnen Vorgehensschritten niedergelegten Ablaufpläne, Algorithmen etc, sowie Listings (vor allem die neuesten) mit Datumsangaben und Variablenbezeichnungen.

3.21.2 Fehlersuche, Fehlerabfangung

3.21.2.1 Programmablauf, TRACE

Vielfach treten in Programmen Fehler auf, die nicht sofort bemerkt werden, sondern erst nach einiger Zeit sichtbar werden. In solchen Fällen ist es häufig schwierig, herauszufinden, durch welche Zeilen das Programm gelaufen ist. Dazu dient der Befehl TRACE; nach der Eingabe von TRACE wird bei der Ausführung jedes Programmschritts ein Nummernzeichen (#) und die momentan aktuelle Programmzeile ausgegeben:

```
5  A = 2
10 FOR I = 1 TO 3: A = A ↑ 2 : NEXT I
20 PRINT A
TRACE
RUN

#5 #10 #10 #10 #10 #10 #10 #10 #10 #10 #20
256
```

TRACE hat allerdings einige Nachteile:
TRACE arbeitet nicht mit DOS 3.3, Routinen für den Filezugriff können unter DOS 3.3 nicht mit TRACE getestet werden.
Durch die Darstellung der Zeilennummern werden die vom Programm ausgegebenen Daten stark gestört; Ausgaben in Tabellenform sind dann nicht mehr sonderlich übersichtlich.
Programmschleifen, die sehr oft durchlaufen werden, dauern wegen der großen Zahl darzustellender Zeilennummern sehr lange. Dabei laufen eventuell auf dem Bildschirm vorhandene Daten nach oben weg.
Bei der Ausgaben von Zeilennummern über einen Drucker sind möglichst leise und schnelle Geräte von Vorteil, sonst werden Nerven und Geduld stark beansprucht.
Die Zeilennummern werden nur im Textmodus ausgegeben, nicht im Grafikmodus.

Die beschriebenen Problem können reduziert werden, wenn der TRACE Befehl kurz vor dem Ort des Fehlers ins Programm geschrieben wird. Leider ist der Fehlerort in den seltensten Fällen genau bekannt, und gerade in diesen Fällen ist TRACE keine besondere Hilfe:

```
5  A = 2
10 FOR I = 1 TO 3: A = A ↑ 2 : NEXT I
15 TRACE
20 PRINT A
RUN

#20
256
```

TRACE wird mit dem Befehl NOTRACE neutralisiert.

3.21.2.2 Programme unterbrechen, STOP, CONT

Der Befehl STOP hält an einer beliebigen Stelle des Programms die Ausführung an und bringt eine Meldung:

```
5  A = 2
10 FOR I = 1 TO 3: A = A ↑ 2 : NEXT I
20 PRINT A
25 STOP
30 PRINT : PRINT "Ende des Programms"
RUN

256
BREAK IN 25

PRINT I
4
CONT

Ende des Programms
```

Nach der Meldung „BREAK IN Zeilennummer“ können beliebige Abfragen durchgeführt werden oder Variablen gesetzt oder geändert werden. Mit dem Befehl CONT (von Continue) wird das Programm an der Stelle der Unterbrechung fortgesetzt. In manchen Fällen (z.B. bei Zugriff auf die Diskette o.ä.) ist

eine Fortsetzung des Programms nicht möglich (CAN'T CONTINUE ERROR).

3.21.2.3 Hilfsroutinen

Die Variablenabfrage während des Programmlaufs kann auch automatisiert werden durch die Einführung zusätzlicher PRINT Befehle, die nach den Tests wieder gelöscht werden:

```
5  A = 2
10 FOR I = 1 TO 3: A = A ↑ 2 : NEXT I
20 PRINT A
25 ZN = 25: GOSUB 60000
30 PRINT : PRINT "Ende des Programms"
60000 PRINT "Zeile " ; ZN ; ": I=" ; I ; "   A=" ; A : RETURN
RUN
```

```
256
Zeile 25: I=3          A=256
Ende des Programms
```

Diese zusätzlichen PRINT Befehle werden am besten wieder in einer Subroutine (mit sehr hoher Zeilennummer) durchgeführt. Bei dieser Methode muß das eigentliche Programm immer den END Befehl enthalten, damit nicht nach dem Ende des eigentlichen Programms noch die Testroutinen durchgeführt werden. Die Subroutinen können nach dem Ende der Programmtests leicht mit DEL beseitigt werden.
Mit der Variablen ZN wird die Zeilennummer in die Subroutine übertragen (Übergabeparameter), damit bei mehrfacher Verwendung der Subroutine klar ist, an welchen Stellen des Programms die Variablen I und A die angegebenen Werte hatten. Wenn der zusätzliche GOSUB 60000 nicht in einer sehr oft durchgeführten Schleife liegt, kann man durch einen zusätzlichen GET- oder INPUT-Befehl in der Subroutine die Programmausführung vorübergehend anhalten und überprüfen, ob die in der PRINT-Routine angegebenen Werte sinnvoll sind oder nicht:

```
60000 PRINT "Zeile "; ZN; ": I="; I;"  A=" ; A: GET X$: RETURN
```

3.21.2.4 Filezugriff kontrollieren

Bei Routinen für den Filezugriff auf Disketten muß man zwischen den Betriebsystemen DOS 3.3 und ProDOS unterscheiden. Unter DOS 3.3 besteht mit den DOS-Befehlen MON und NOMON die Möglichkeit der Überwachung des Datenstroms von und zur Diskette.

MON C,I,O

stellt unter DOS 3.3 aus einem Programm heraus alle Informationen eines Filezugriffs dar. Dabei muß mindestens eines der drei Argumente C,I oder O angegeben werden. Bei Verwendung von C werden die Anweisungen (also READ, INPUT, CLOSE etc) dargestellt, bei I der vom File gelesene Datenstrom, und bei O der auf die Diskette ausgegebene Datenstrom. Innerhalb von Programmen muß MON C,I,O in einer PRINT Anweisung nach CONTROL-D geschrieben werden.
Mit NOMON wird der MON Befehl aufgehoben. Unter ProDOS sind die Befehle nicht zulässig (dafür ist aber TRACE möglich).

3.21.2.5 Programme verzögern, anhalten

In vielen Fällen (vor allem bei Grafik) kann der Programmablauf verfolgt werden, wenn die Arbeitsgeschwindigkeit des Computers reduziert wird. Das geschieht mit dem Befehl SPEED= . SPEED= kann Werte zwischen 0 und 255 annehmen (255 stellt sich beim Einschalten ein). Der Computer zieht diese Zahl von 255 ab und durchläuft entsprechend dem Ergebnis nach jedem Programmschritt eine Warteschleife. Mit

SPEED= 0

ist der Computer am langsamsten, mit 255 am schnellsten. Testen mit SPEED= hat den Vorteil, daß das Programm nicht verändert wird und originalgetreu abläuft.

Zu Prüfzwecken wird mit dem Befehl CONTROL-S der Programmablauf angehalten. Bei erneutem CONTROL-S wird das Programm fortgesetzt. Diese Zwangspausenmethode stört den Programmablauf ansonsten nicht. Das Verfahren kann beispielsweise auch bei CATALOGs angewendet werden, um die laufende Darstellung der Files auf dem Bildschirm vorübergehend anzuhalten.

3.21.2.6 Fehlerabfangung

Applesoft Basic ermöglicht mit dem Befehl ONERR GOTO 'Zeilennummer' die Fehlerabfangung:

```
10  ONERR  GOTO 1000
```

Zeile 10 hat keine Wirkung, solange kein Programmfehler auftritt. Beim ersten Fehler im Programm erfolgt ein Sprung in Zeile 1000. Dort ist eine Fehlerbehandlungsroutine installiert:

```
1000  REM  ONERR ROUTINE
1010  F =  PEEK (222): REM  FEHLERNUMMER
1020  Z =  PEEK (218) + PEEK(219) * 256 : REM   FEHLERZEILE
1030  IF F = 133 THEN  PRINT "Unzulässige Division durch
      Null in Zeile ";Z: POKE 216,0: GOTO 100: REM  Hauptmenue
1040  PRINT "Fehler Nr.";F;" in Zeile ";Z: POKE 216,0:
      GOTO 100: REM  Hauptmenue
```

In Zeile 1010 wird durch PEEK(222) die Fehlernummer festgestellt. Fehler Nummer 16 beispielsweise ist der „SYNTAX ERROR". Die zu den Nummern zugehörigen Fehlerbezeichnungen sind im Anhang dargestellt.

Durch PEEK(218) + PEEK(219) * 256 wird die Zeile ermittelt, in der der Fehler auftrat. Anschließend kann dem Benutzer ein detaillierter Hinweis auf den Fehler gegeben werden (Zeile 1030), oder wie in Zeile 1040 eine allgemein gebräuchliche Fehlermeldung formuliert werden.
In beiden Zeilen wird durch POKE 216,0 die ONERR GOTO Anweisung außer Kraft gesetzt. Der Grund dafür ist folgender:
Die Methode der Fehlerabfangung hat den Nachteil, daß bei Fehlern innerhalb der Fehlerbehandlung das Programm in eine endlose Schleife gerät. Die Außerkraftsetzung der ONERR Anweisung mit POKE 216,0 verhindert das. Damit ist zwar wieder ein Programmabbruch aufgrund eines Programmfehlers möglich, dies ist aber immer noch besser als eine endlose Schleife.

Basic stellt mit dem Befehl RESUME eine umfassende Möglichkeit zur Fehlerbehandlung zur Verfügung; RESUME springt an die Stelle zurück, an der der Fehler auftrat. Wegen der oben genannten Gefahr der Endlosschleife ist es zwingend notwendig, alle möglichen auftretenden Fehler in der Fehlerabfangung vor der Ausführung des RESUME Befehls zu eliminieren, z.B. Benutzerhinweis geben bei „TYPE MISMATCH ERROR". In manchen Fällen ist dies aber nicht möglich (z.B. bei REDIMENSIONED ARRAY). Aus diesen Gründen sollte sich die umfassende Fehlerbehandlung auf eindeutig kritische Pro-

grammsequenzen beschränken, z.B. auf die Prüfung beim Lesen von Disketten-Textfiles auf END OF DATA Fehler (Nr. 5) mit anschließendem CLOSE des Files. In diesem Fall muß die Fehlerroutine mit POKE 216,0 wieder aufgehoben werden, sonst wird beim nächsten Fehler (der vielleicht gar nichts mit Disketten zu tun hat) versucht, ein File zu schließen.

3.21.3 Hilfsprogramme

Hilfsprogramme stellen dem Benutzer zusätzliche Informationen und Möglichkeiten zur Verfügung, die beim Programmablauf nicht notwendig sind, aber bei der Programmentwicklung und in der Testphase sehr nützlich sein können. Ein solches Hilfsprogramm ist das Programm RENUMBER unter DOS 3.3. RENUMBER stellt neue Befehle zur Verfügung, die mit dem &-Zeichen beginnen. Diese Befehle sind

&
&HOLD
&MERGE

Das & Zeichen führt eine Neunumerierung des im RAM vorhandenen Basic-Programms durch. Die Neunumerierung erfolgt in 10er Schritten beginnend ab Zeile 10, die neuen Zeilennummern heißen dann 10, 20, 30 usw. Gleichzeitig werden die Sprungzeilennummern hinter GOTO- und GOSUB-Befehlen geändert. Die Neunumerierung läßt sich bezüglich Zeilennummernintervall etc. ändern.

Mit den Befehlen &HOLD und &MERGE können mehrere Programme oder Programmteile zusammengesetzt werden. Damit können bestimmte, in vielen Programmen vorkommende Teile (z.B. Eingabesubroutinen) separat gespeichert werden und in jedes neue Programm übernommen werden. Der Befehl &HOLD sichert ein im RAM vorhandenes Programm gegen Löschen und Listen; anschließend kann ein weiteres Programm mit LOAD von der Diskette gelesen werden.

```
10  REM - Hauptprogramm
20  TT$ = "ADRESSVERZEICHNIS":GOSUB 100: REM Titel
30  INPUT "Bitte Vornamen eingeben: ";VN$
.
.
.
99 END : REM Ende Hauptprogramm
```

&HOLD

Mit dem Befehl &HOLD wird das bestehende Programm vor dem Überschreiben durch ein nachfolgendes LOAD geschützt.

LOAD SUB.TITEL

LIST

```
100 REM - Titelroutine
110 PRINT CHR$(4);"PR#3"
120 PRINT : PRINT "Copyright (c) 1985 by Charlie Chaplin"
130 INVERSE : PRINT HTAB 10 ; TT$
140 POKE 34,5
150 RETURN
```

&MERGE

LIST

```
10  REM - Hauptprogramm
20  TT$ = "ADRESSVERZEICHNIS":GOSUB 100: REM Titel
30  INPUT"Bitte Vornamen eingeben: ";VN$
.
.
99  END : REM Ende Hauptprogramm
100 REM - Titelroutine
110 PRINT CHR$(4);"PR#3"
120 PRINT : PRINT "Copyright (c) 1985 by Charlie Chaplin"
130 INVERSE : PRINT HTAB 10 ; TT$
140 POKE 34,5
150 RETURN
```

Mit dem Befehl &MERGE werden diese beiden Programme zusammengesetzt. Enthalten beide Programm(teile) teilweise gleiche Zeilennummern, so werden die Zeilen von dem zuerst im RAM vorhandenen Programm gelöscht.
Unter ProDOS arbeitet das Programm RENUMBER nicht. Für DOS 3.3 und ProDOS wird das Programm APA angeboten, das eine Reihe von Editorbefehlen bietet (ähnlich RENUMBER).
Im Handel werden spezielle Editorprogramme für den Apple angeboten, welche die Eingabe und Untersuchung von Programmen vereinfachen.

Die Firma APPLE bietet für APPLESOFT-Programmierer einen APPLESOFT/DOS Programmer's Tool Kit an. Dieser Tool Kit (= Werkzeugkiste) bietet eine Reihe von Hilfsprogrammen für verschiedene Zwecke. Das schon vorgestellte Programm APA beispielsweise ermöglicht Programme neu zu

numerieren, zu komprimieren (entfernen der Remarks), erstellt eine Variablenliste etc. Unter anderem ist außerdem ein komfortabler Editor und ein Schriftsatzgenerator vorhanden.

3.21.4 Programme mit Textverarbeitung ändern

Alle APPLESOFT-Programme können mit Textverarbeitungsprogrammen geändert werden. Das Verfahren ist aufwendiger als die Programmänderung mit dem „ESCAPE-Editor", weil dazu „Reboots" und etwas „Diskhandling" nötigt ist. Diese Technik wird deshalb besser nur bei größeren Programmen verwendet. Es ist sinvoll, in diesem Fall möglichst viele Änderungen im Programm auf einmal durchzuführen. Für kleine Programme, die zu Testzwecken ständig geändert und ausgeführt werden, empfiehlt sich der übliche „ESCAPE-Editor".

APPLESOFT-Programme, die unter DOS 3.3 laufen, werden am besten mit einem Textverarbeitungsprogramm geändert, welches ebenfalls unter DOS 3.3 läuft. Ein solches Programm ist z.B. APPLEWRITER //e. Dieses Programm ist für den Apple //e geschrieben worden, läuft aber (mit kleinen Einschränkungen) auch auf dem Apple //c.
BASIC-Programme, die in ProDOS geschrieben worden sind, können beispielsweise mit dem Programm APPLEWORKS editiert werden. Das APPLEWORKS-Programm läuft unter ProDOS.
Zwar können alle Files mit Hilfe der ProDOS „System Dienstprogramme" von DOS 3.3 Files in ProDOS Files und umgekehrt verwandelt werden, diese Technik macht allerdings die Programmänderung mit Textverarbeitung noch umständlicher.

Grundsätzlich funktioniert diese Technik der Programmänderung mit allen Textverarbeitungssytemen, welche Text- oder ASCII-Files unter dem gewünschten Betriebssystem lesen und beschreiben können.
Die Vorgehensweise besteht grob aus folgenden Schritten:

1. Umwandeln des APPLESOFT BASIC Programms in ein Textfile (ASCII-File)
2. Starten des Textverarbeitungsprogramms
3. Einlesen des Textfiles (ASCII-File) mit dem Programmtext in das Textverarbeitungsprogramm
4. Ändern des Programmtextes mit dem Textverarbeitungsprogramm
5. Ausgeben (drucken) des geänderten Textes auf eine Textfile (ASCII-File)

6. Starten der Systemdiskette (ProDOS oder DOS 3.3)
7. Umwandeln des geänderten Textfiles in ein Programm mit dem Befehl EXEC
8. Starten oder Abspeichern des geänderten Programms

Wird ein Programm gleich von Anfang an mit Textverarbeitung geschrieben, dann erübrigen sich natürlich die Schritte 1 und 3.

Die Umwandlung eines APPLESOFT BASIC-Programms in ein Textfile geschieht entweder von Hand oder durch eine kurzes EXEC-File. Ein EXEC-File ist ein Textfile, welches Befehle (BASIC oder Betriebssystem) in Textform enthält. Dieses File kann mit dem Befehl EXEC (von Execute = Ausführen) gestartet werden, ohne daß ein schon im RAM vorhandenes APPLESOFT Programm zerstört wird.

Das unten dargestellte Programm erzeugt auf der Diskette ein EXEC-File, also ein Textfile. Dieses EXEC File heißt PROG.TEXT.

```
10  REM  - Dieses Programm erzeugt ein EXEC File, welches
     ein Programm in ein Textfile schreibt.
15  REM  - Das EXEC File wird gestartet durch "EXEC PROG.TEXT"
20 D$ =  CHR$ (4):A$ =  CHR$ (34): REM   - CONTROL-D, (")
30  PRINT D$;"OPEN PROG.TEXT"
40  PRINT D$;"WRITE PROG.TEXT"
50  PRINT "57777 PRINT CHR$(4);";A$;"OPEN PROGRAMMLISTING";A$
60  PRINT "57778 PRINT CHR$(4);";A$;"WRITE PROGRAMMLISTING";A$
70  PRINT "57779 LIST,57776": PRINT "57780 LIST 57782,"
80  PRINT "57781 PRINT CHR$(4);";A$;"CLOSE";A$
90  PRINT "57782 RUN 57777'
100  PRINT "DEL 57777,57781'
110  PRINT D$;"CLOSE"
120  END
```

Ist das EXEC-File PROG.TEXT einmal erzeugt, so wird das oben dargestellte Programm nicht mehr benötigt.
Das zu ändernde Programm wird nun mit LOAD Filename (hier als Beispiel 'HAUPTPROGRAMM') ins RAM gelesen.

LOAD HAUPTPROGRAMM

Anschließend wird das File PROG.TEXT ausgeführt.

EXEC PROG.TEXT

PROG.TEXT erzeugt im Hauptprogramm die Zeilen 57777 bis 57781, führt diese aus, und löscht sie wieder. Bei der Ausführung der Programmzeilen 57777 bis 57781 wird ein Listing des HAUPTPROGRAMMS (ohne die von

PROG.TEXT erzeugten Zeilennummern) auf ein Text-File namens PROGRAMMLISTING auf der Diskette im Laufwerk geschrieben. Das HAUPTPROGRAMM ist unverändert noch im RAM vorhanden.

Wem diese Vorgehensweise zu kompliziert ist, kann das Textfile mit dem Inhalt des Basic Programms direkt erzeugen:
Dazu wird das im RAM Speicher vorhandene Basic Programm um eine Zeile erweitert, z.B:

```
64000 PRINT CHR$(4);"OPEN PROGRAMMLISTING": PRINT CHR$(4);
      "WRITE PROGRAMMLISTING": LIST, 63999: PRINT CHR$(4);
      "CLOSE" : END
```

Die Programmzeile muß entweder ganz am Anfang des Programms stehen (z.B. Zeile 0), oder (wie hier) am Ende. Sie wird mit

GOTO 64000

aufgerufen, eröffnet ein File mit Namen 'PROGRAMMLISTING', das anschließend das Programmlisting enthält, und schließt es wieder.

Jetzt kann mit einem Textverarbeitungssystem das Textfile PROGRAMMLISTING gelesen werden; bei DOS 3.3 und Textverarbeitungsprogramm APPLEWRITER mit Control-L und Filename (PROGRAMMLISTING), bei ProDOS und APPLEWORKS mit Hauptauswahl 1, Unterauswahl 3 (2. von einer ASCII Datei). Bei APPLEWORKS muß der gesamte Pfadname des Files eingegeben werden, also auch der Diskettenname.

Die Textform des Programms kann nun nach den Möglichkeiten des Textverarbeitungsprogramms verändert werden.

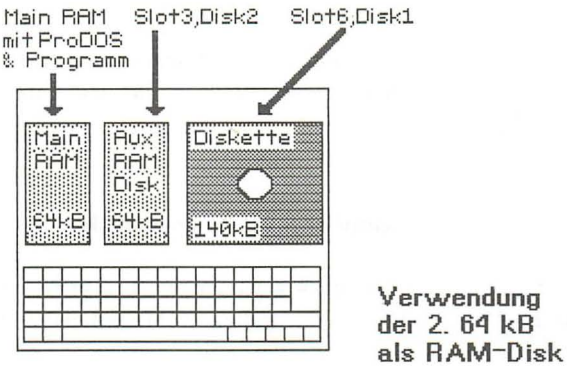
Anschließend wird das Programm wieder als Text ausgegeben. Bei APPLEWRITER mit dem Befehl Control-P (bzw. Control-D) und Filename; bei APPLEWORKS mit dem Druckbefehl „in eine ASCII Datei auf Diskette". Um das alte unveränderte Textfile PROGRAMMTEXT nicht zu zerstören, wird am besten für den abzuspeichernden, geänderten Text ein anderer Name gewählt.

Nach dem Reboot mit der Sytemdiskette (DOS 3.3 bzw. ProDOS) kann das geänderte Textfile mit EXEC aufgerufen werden, anschließend mit RUN ausgeführt oder mit SAVE als Programm gespeichert werden.

3.22 Die Verwendung der RAM-Disk

Der Apple //c besitzt 128 kB RAM, aufgeteilt in 2 mal 64 kB. Der Prozessor des Apple //c kann wegen des 16 Bit umfassenden Adressraums nur 64 kB auf einmal „sehen". Zwar kann mit Hilfe besonderer Softswitches (Page $C000 bis $C0FF) der Arbeitsbereich des Prozessors umgeschaltet werden, doch ist dieses Vorgehen aufwendig und muß gut geplant werden.

Das Betriebssystem ProDOS enthält jedoch einen RAM-Disk-Driver, mit dessen Hilfe ca. 60 kB der oberen 64 kB Bank als 'Diskette' genutzt werden können.



Der Vorteil einer solchen RAM-Disk liegt in der, verglichen mit physischen Disketten, hohen Zugriffsgeschwindigkeit. Die Informationen der RAM-Disk gehen beim Ausschalten des Computers natürlich verloren.
Die RAM-Disk wird als Slot 3, Disk 2 angesprochen:

```
CAT ,S3,D2

/RAM
NAME            TYPE   BLOCKS  MODIFIED

BLOCKS FREE:     121   BLOCKS USED:   7
```

Auch kann der Prefix auf die RAM-Disk gesetzt werden:

PREFIX/RAM

Die RAM-Disk wird von ProDOS genau gleich behandelt wie eine Diskette:

SAVE /RAM/Filename

LOAD /RAM/Filename

RENAME /RAM/File1,File2

etc.

Wegen der hohen Zugriffsgeschwindigkeit bietet sich deshalb die RAM-Disk als Zwischenspeicher für Daten und Programmteile an, die in den freien 36 kB des Main Memory (Hauptspeichers) keinen Platz haben.

3.22.1 RAM-Disk als Zwischenspeicher für Programmteile

Um Speicherplatz im Main Memory zu gewinnen, wird ein großes Benutzerprogramm in einzelne Programmteile aufgespalten. Zum Starten des Programmpakets wird ein Initialisierungsteil verwendet, z.B.:

RUN PROG.INIT

```
10 REM - Initialisierungsroutine
20 REM - erzeugt EXEC File, welches Programmteile in
       die RAM-Disk lädt.
30 D$ = CHR$(4)
40 PRINT D$;"OPEN /RAM/EXECFILE"
50 PRINT D$;"WRITE /RAM/EXECFILE"
60 PRINT "LOAD PROG.TEIL1"
70 PRINT "SAVE /RAM/PROG.TEIL1"
80 PRINT "LOAD PROG.TEIL2"
90 PRINT "SAVE /RAM/PROG.TEIL2"
100 PRINT "LOAD PROG.TEIL3"
110 PRINT "SAVE /RAM/PROG.TEIL3"
120 PRINT "PREFIX/RAM"
130 PRINT "RUN PROG.TEIL1"
140 PRINT D$;"CLOSE"
150 PRINT D$;"EXEC /RAM/EXECFILE"
```

Das Programm erzeugt auf der RAM Disk ein Textfile, welches anschließend EXECuted (ausgeführt) wird. Das EXEC-File (Textfile) liest von der Programmdiskette in Laufwerk 1, auf der sich auch das Startprogramm PROG.INIT befindet, 3 Programmteile (je nach Bedarf lassen sich eine beliebige Anzahl Programmteile laden);

PROG.TEIL1
PROG.TEIL2
PROG.TEIL3

Diese Teile werden in die RAM-Disk kopiert. Anschließend wird der erste Teil (PROG.TEIL1) ausgeführt. PROG.TEIL1 muß alle Dimensionierungen ausführen. Die weiteren Programmteile werden je nach Bedarf durch den „-CHAIN" Befehl ausgeführt. Der „CHAIN" Befehl löscht das alte Programm, liest das angegebene Programm von der RAM-Disk und führt es aus. Die vom alten Programm definierten Variablen werden nicht gelöscht.

Zur Verdeutlichung sind hier drei solche Programmteile dargestellt, die sich nacheinander aufrufen. Zur Kontrolle der Variablen wird die Zahl A in jedem Programmteil dargestellt.

```
10 REM - PROGRAMMTEIL 1
20 PRINT "PROG.TEIL1"
30 DIM A$(20) , B%(20,4) : REM - alle Dimensionierungen
40 A = 12.2
50 PRINT "A=";A
60 PRINT CHR$(4);"CHAIN PROG.TEIL2"

10 REM - PROGRAMMTEIL 2
20 PRINT "PROG.TEIL2"
30 PRINT "A=";A
40 PRINT CHR$(4);"CHAIN PROG.TEIL3"

10 REM - PROGRAMMTEIL 3
20 PRINT "PROG.TEIL3"
30 PRINT "A=";A
```

Anmerkung: Derjenige Programmteil, der die Dimensionierung (DIM) ausführt, darf nur einmal aufgerufen werden. Eine zweite Ausführung würde einen „REDIM'D ARRAY ERROR" hervorrufen.

Die Methode hat zusätzlich den Vorteil, daß häufig wiederkehrende Teile in allen Programmen verwendet werden können (Modultechnik). Dazu werden sie lediglich im Startprogramm (hier PROG.INIT) aufgerufen.

Die Trennung des Programms in einzelne Teile geschieht zweckmäßigerweise bei Benutzereingaben oder anderen nicht zeitkritischen Sequenzen. In oft durchlaufenen Zeilen ist die Trennung wegen des (gegenüber anderen Befehlen) relativ langsamen Chaining ungünstig.

3.22.2 RAM-Disk als Zwischenspeicher für Daten

Kritische Datenmengen entstehen im allgemeinen bei dimensionierten Variablen oder bei Grafik. Bei Verwendung der RAM Disk werden diese Daten wie bei Disketten in Files gespeichert und gelesen, je nach Bedarf fileweise, in Gruppen (z.B. 100 Strings) oder einzeln.

Für numerische Berechnungen größeren Umfanges bei gleichzeitig großen Datenmengen wie Finite Element Rechnungen ist die hier beschriebene Zwischenspeicherung wegen des Zeitbedarfs ungeeignet.
Die Speicherung von Grafik oder Strings (in der Regel eindimensional) ist dagegen gut möglich.
Das Beispiel zeigt den Zugriff auf eine vom Benutzer gewünschte Zahl in einem Feld von 1000 Gleitkommazahlen:

```
0 REM
        - Abkürzungen definieren

10 W$ =  CHR$ (4) + "WRITE FILE,R"
20 R$ =  CHR$ (4) + "READ FILE,R"
30 O$ =  CHR$ (4) + "OPEN FILE,L15"
40 C$ =  CHR$ (4) + "CLOSE"
50 PRINT CHR$(4)"PREFIX/RAM/"
190  REM
        - File mit 1000 Zufallszahlen füllen

200  PRINT O$
210  FOR I = 1 TO 1000
220  PRINT W$;I: PRINT RND(4)
230  NEXT I
240  PRINT C$
300  REM
        - Benutzerdefinerte Zahl aus dem File holen

310  PRINT O$
320  INPUT "Welche Zahl (1 bis 1000) wird gewünscht ? ";X
330  PRINT R$;X: INPUT A$
340  PRINT A$,I
350  PRINT  CHR$ (4)
360  GOTO 320
```

An dem Beispiel wird sichtbar, daß die Zugriffsgeschwindigkeit für diese Art von Benutzereingaben vollständig ausreicht. Für Sortierprogramme oder andere Zwecke, wo sehr häufiger Datenzugriff notwendig wird, ist es günstiger, Daten blockweise zu lesen bzw. zu schreiben.

3.22.3 Die RAM-Disk als Zwischenspeicher für Hires Grafiken

Folgendes Programm kann in Grafikprogramme eingebunden werden (siehe z.B. Programmierung der Maus), um hochauflösende Grafik (HGR) zwischenzuspeichern:

```
100 INPUT"Filename für das Bild:";F$
110 PRINT CHR$(4);"BSAVE /RAM/";F$;",A$2000,E$3FFB"
```

Analog wird mit BLOAD ein File von der RAM-Disk gelesen.

Anmerkung: Die Formulierung BSAVE Filename ,Anfangsadresse, Endadresse ist nur unter ProDOS zulässig. Statt voller 8 Pages ($2000) werden hier 5 Byte weniger gespeichert. Damit läßt sich ein Sektor auf der Diskette (bzw. RAM-Disk) sparen (33 Sektoren statt 34 Sektoren pro File), ohne daß Information fehlt.

Der Inhalt der RAM-Disk geht beim Ausschalten des Computers verloren. Alle wichtigen Files müssen also vorher auf Diskette kopiert werden. Dazu werden entweder eigene Programme geschrieben oder das Programm „FILER“ auf der ProDOS USERS DISK verwendet.

Kapitel 4

ANDERE PROGRAMMIERSPRACHEN

4.1 Ein Überblick

Außer der Sprache Applesoft Basic „versteht" der Apple //c noch eine ganze Reihe weiterer Sprachen. Bis auf Assembler, welches wie Basic ebenfalls schon auf dem Apple //c (zumindest zum Teil) installiert ist, müssen alle diese Sprachen von Disketten eingelesen werden.

Die Programmiersprachen haben sehr unterschiedlichen Charakter; alle dienen der Übersetzung des vom Programmierer geschriebenen Programmtextes in den Maschinencode, also eine reine Kette von Zahlen. Die Art der Übersetzung und der Wortschatz, d.h. die zulässigen Befehle und Befehlsanordnungen, sind in jeder Sprache anders. Deshalb sind sie auch für bestimmte Anwendungen unterschiedlich gut geeignet.

Für eine Programmiersprache ist es wesentlich, ob sie „interpretiert" wird oder „compiliert" ist:
Ein Interpreter „übersetzt" den momentan aktuellen Befehl eines Programmes von der Textform während der Ausführung des Programmes in einen Maschinenbefehl. Interpretierte Sprachen sind BASIC, LOGO und PILOT.
Compilersprachen übersetzen den Programmtext nach der Eingabe vollständig in Maschinencode. Dieser wird anschließend ausgeführt.
Compilierte Programme haben den Vorteil der wesentlich höheren Arbeitsgeschwindigkeit. Außerdem wird der Programmtext zwangsläufig bei der Compilierung auf Syntaxfehler geprüft. Solche Fehler treten bei der Ausführung dann nicht mehr auf. Die Compilierung braucht allerdings etwas Zeit (je nach Sprache oder Dialekt unterschiedlich viel). Daher ist jede, wenn auch noch so kleine Änderung eines Programmes deutlich aufwendiger als beispielsweise in BASIC.

Die verschiedenen Programmiersprachen sind bezüglich Interpeter oder Compilermethode nicht festgelegt. Für BASIC beispielsweise sind verschiedene Compiler erhältlich, auf manchen Computern wird PASCAL als Interpreter statt als Compilersprache angeboten. Eine Sonderstellung nimmt die Sprache FORTH ein, die sowohl Interpreter als auch Compiler besitzt.

Für die Computer der Apple // Familie sind vor allem folgende Sprachen erhältlich:

APPLESOFT BASIC	
ASSEMBLER	
COBOL	
FORTH	(verschiedene Versionen)
FORTRAN	(verschiedene Versionen)
LOGO	(verschiedene Versionen)
MODULA2	
PASCAL	
SuperPILOT	

Diese Sprachen sind alle für den Apple //e bzw. den Apple // (plus) geschrieben worden. Es ist möglich, daß bei verschiedene Sprachen (oder auch Dialekten, wenn mehrere Versionen existieren) manche Befehle oder Tasten auf dem Apple //c nicht funktionieren. Die meisten Sprachen arbeiten aber ohne Probleme.

Die Programmiersprachen lassen sich wie folgt charakterisieren:

ASSEMBLER, manchmal auch Maschinensprache genannt, ist eine sehr systemnahe und äußerst schnelle Sprache. Man kann den Computer praktisch beliebig programmieren, allerdings ist die Programmierung sehr aufwendig. Die Programme sind sehr schwer lesbar.

COBOL ist eine verbreitete Sprache, die vor allem für kommerzielle betriebliche Anwendungen verwendet wird.

FORTH ist eine junge Sprache, die sehr viele Eigenschaften anderer Sprachen zusammenfaßt. Sie ist schwer zu erlernen, schöpft aber die Möglichkeiten eines Computers am besten aus.

FORTRAN ist die älteste höhere Programmiersprache. In ihrem Aufbau ist sie etwa der Sprache PASCAL ähnlich. Auf dem Apple wird sie selten angewendet.

LOGO und **SuperPILOT** sind Lernsprachen (vor allem für Kinder). Ihre einfache Syntax erlaubt den spielerischen Umgang mit dem Computer.

PASCAL ist eine stark strukturierte Sprache, die weit verbreitet ist. Die Programmierung in PASCAL erfordert zu Anfang einigen Mehraufwand gegenüber BASIC.

MODULA 2 ist eine Weiterentwicklung von PASCAL.

In den nachfolgenden Abschnitten werden die Sprachen PASCAL, FORTH und ASSEMBLER kurz dargestellt. Die Ausführungen sind keine Lernunterlagen für diese Sprachen. Sie sollen dem Leser den Charakter der Sprachen verdeutlichen und ihm eine eventuelle Entscheidung für oder gegen die Anwendung dieser Sprachen ermöglichen.
Die Abhandlung über Assembler befähigt den Leser zur begrenzten Programmierung und gibt ihm die Grundlagen für die Eingabe und Verwendung kurzer Maschinenprogramme, wie sie in Fachzeitschriften zu finden sind.

4.2 PASCAL

PASCAL ist eine Compilersprache. Sie arbeitet mit einem eigenen Betriebssystem, dem UCSD-P System (UCSD = University of California, San Diego). Das UCSD-P System unterscheidet sich sehr stark von den bisher verwendeten DOS-Betriebssystemen. UCSD PASCAL läuft auch auf vielen anderen Computern, da es nicht speziell für den Apple entwickelt worden ist. Die PASCAL-Programme können also leicht auf andere Computer übertragen werden.

Das UCSD-P System ist in Bezug auf den Speicherbedarf deutlich größer als das DOS. Aus diesem Grund ist es auf mehrere Disketten verteilt. Im Apple ist immer nur ein Teil des Betriebssystems vorhanden. UCSD-Pascal wurde auf Vorgängern des Apple //c mit 64 kB RAM entwickelt. Aus diesem Grund verwendet das ältere PASCAL bisher auch nur die ersten 64kB des Apple //c. Die neuere PASCAL Version 1.2 nutzt den gesamten Speicherbereich des APPLE //c.
UCSD-PASCAL greift relativ häufig auf Diskettendaten zu. Zweckmäßigerweise werden für PASCAL zwei Diskettenlaufwerke verwendet, da mit nur einem Laufwerk häufig Disketten gewechselt werden müssen und die mögliche Programmgröße stark eingeschränkt ist.

Ist der APPLE //c mit einem Z80-Zusatzprozessor versehen, so kann das TURBO-PASCAL benutzt werden. Dieses PASCAL arbeitet sehr schnell und einfach. Wegen der notwendigen Hardwareänderung soll hier auf Unterschiede zum UCSD PASCAL nicht eingegangen werden.

UCSD PASCAL wird auf vier Disketten geliefert. Auf diesen Disketten sind eine Reihe von Files vorhanden, die im Wesentlichen die folgenden Aufgaben erfüllen:

- Ein Editor, mit dem Programme eingegeben und geändert werden können.
- Ein Compiler, der die Textform des Programms in ausführbaren p-Code übersetzt,
- Ein Filer, der die Fileverwaltung (kopieren, löschen etc.) besorgt.
- Ein Assembler
- Ein Linker, der verschiedene Programmteile und Standardroutinen zusammenhängt.

Daneben sind weitere Systemfiles vorhanden, z.B. eine Library, die verschiedene ausformulierte Unterprogramme enthält. Die Diskette PASCAL3: enthält ein Formatierungsprogramm und mehrere Beispiele von kurzen PASCAL-Programmen.

Ein Programm in PASCAL wird in folgenden groben Schritten erstellt:

1. Eingabe des Programms mit dem EDITOR.
2. Speichern des eingebenen Programmtextes auf Diskette in „Filename.TEXT".
3. Compilierung (Übersetzung) des Textes mit dem COMPILER, gleichzeitig speichern des compilierten Codes auf Diskette unter „Filename.CODE".
4. Ausführen des Programmcodes.

Jede Änderung des Programms geschieht im EDITOR (einem Textverarbeitungsprogramm) und bedingt vor der erstmaligen Ausführung die erneute Compilierung.

4.2.1 PASCAL-Programme

PASCAL zwingt den Benutzer zur strukturierten Programmierung. PASCAL-Programme enthalten keine Zeilennummern. Aus diesem Grund ist die Basicspezifische Spagettistruktur mit vielen GOTO's nicht möglich.

PASCAL unterstützt die Aufteilung eines Programmes in Unterprogramme, auch Prozeduren (Procedures) genannt. In jeder dieser Prozeduren können

eigene Variablennamen verwendet werden; eine Variable A kann in einer Prozedur eine Integerzahl sein, in einer anderen einen String darstellen. Die in der Prozedur verwendeten Variablen und ihr Typ werden zu Beginn definiert. Eine typische PASCAL-Prozedur sieht beispielsweise so aus:

```
PROCEDURE AUSGABE (X,Y,Z: INTEGER);
VAR  A,B,C: STRING;
BEGIN
  A:= 'Das Produkt aus ';
  B:= ' und ';
  C:= ' ist ';
  WRITELN (A, X, B, Y, C, Z);
END;
```

Die erste Zeile gibt den Prozedurnamen an, hier „AUSGABE", unter dem die Prozedur aufgerufen wird. In der gleichen Zeile stehen (in Klammern) die Übergabeparameter, hier drei Integerzahlen, welche beim Aufruf der Prozedur übergeben werden.
In Zeile zwei sind drei Stringvariablen definiert, welche nur in dieser Prozedur die definierte Bedeutung haben.
Durch den Befehl BEGIN wird die eigentliche Prozedur eröffnet. Darin werden den drei Stringvariablen A,B,C Texte zugeordnet, und mit dem Befehl WRITELN werden die Texte und die Integerzahlen ausgegeben.
Die Prozedur wird mit dem Befehl END abgeschlossen.
Die Strichpunkte an den Zeilenenden geben das Ende eines Befehls wieder; dieser endet in der Regel am Zeilenende. Der Befehl BEGIN hingegen, mit dem die eigentliche Prozedur beginnt, ist erst durch das END in der letzten Zeile abgeschlossen. Deshalb steht nach BEGIN kein Strichpunkt, sondern erst nach dem END.

Ein ganzes PASCAL-Programm besteht aus einer Reihe solcher Prozeduren, die von anderen Prozeduren oder vom Hauptprogramm aufgerufen werden können. Prozeduren können also wie Basic Subroutinen (GOSUB ... RETURN) ineinander geschachtelt werden. Eine Prozedur muß im Programm definiert sein, bevor sie aufgerufen wird, also im Text oberhalb des Aufrufs. Der Aufruf einer Prozedur geschieht durch Nennung ihres Namens. Programmbeispiel:

```
PROGRAM BEISPIEL1;                    Name des Programms

PROCEDURE EINGABE;                    Erste Prozedur
BEGIN
...
END;
```

```
PROCEDURE RECHNUNG (...);                Zweite Prozedur
BEGIN
...
END;

PROCEDURE AUSGABE (...);                 Dritte Prozedur
BEGIN
...
END;

BEGIN (* Hauptprogramm *)                Anfang des Hauptprogramms
  EINGABE;                               Aufruf der Prozeduren
  RECHNUNG;
  AUSGABE;
END.                                     Ende des Hauptprogramms
```

Das Hauptprogramm steht am Schluß; es ist mit einem Kommentar versehen (entpricht in Basic dem REM Befehl).
Das Hauptprogramm ruft nacheinander die Prozeduren EINGABE, RECHNUNG und AUSGABE auf. Diese führen die eigentlichen Befehle aus. Die Prozeduren sind in diesem Beispiel nicht geschachtelt. Die Prozedur AUSGABE könnte aber z.B. die Prozedur RECHNUNG nochmals aufrufen, da diese vor der Prozedur AUSGABE definiert worden ist.

Das Programm wird aus dem Betriebssystem mit dem Befehl X für Execute = Ausführen gestartet.
Die Arbeit mit dem PASCAL-Betriebssystem scheint zunächst etwas umständlich und langwierig, nach der Eingewöhnungszeit ermöglicht PASCAL aber die saubere und effiziente Programmierung. Die Struktur und die Syntax der PASCAL-Programme ist sehr logisch aufgebaut und erleichert das Lesen und Verstehen auch fremder Programme.

4.3 FORTH

Die Sprache FORTH wurde vor etwa 10 Jahren in den USA entwickelt. FORTH unterscheidet sich wesentlich von allen anderen Programmiersprachen. FORTH umfaßt praktisch alles, was ein eingeschalteter Computer an Software enthalten kann:

FORTH ist sowohl
- eine höhere Programmiersprache
- hat Assemblereigenschaften
- ist ein Betriebssystem

- und ein Programmentwicklungssystem.

Die Vielseitigkeit wird im Prinzip ganz einfach erreicht:
Ein Befehl in der Sprache FORTH besteht aus beliebigen anderen Befehlen, die vorher definiert wurden. Dieser neue Befehl kann wiederum verwendet werden, um weitere Befehle zu definieren. Das kann so weit gehen, daß das gewünschte Programm aus einem einzigen Befehl besteht.

In anderen Programmiersprachen, BASIC zum Beispiel, werden Befehle aus dem definierten Grundwortschatz zusammengesetzt. Ein Programm ist dann eine Aneinanderreihung von Befehlen. Beispiel:

```
10 PRINT "Reihe der Quadratzahlen"
20 FOR I=1 TO 10 : PRINT I , I ↑ 2 : NEXT I
30 PRINT "Ende"
40 END
```

Dieses Programm besteht aus mehreren aneinandergereihten Befehlen.

In FORTH kann dieses Programm durch den Befehl „QUADRAT“ definiert werden, wenn man die Unterbefehle, in FORTH auch Worte genannt, vorher definiert hat. Dies geschieht so:

```
: TITEL ."Reihe der Quadratzahlen " CR ;
: RECHNUNG 10 1 DO I . 16 SPACES I I * . CR LOOP ;
: QUADRAT TITEL RECHNUNG CR ."Ende " CR ;
```

Diese Zeilen bedeuten:
Das Wort TITEL schreibt den angegebenen Titel.
Das Wort RECHNUNG führt eine Scheife aus (Grenzen 1 bis 10), in der die Laufvariable I und ihre Quadratzahl angegeben wird.
Das Wort 'QUADRAT' wird gebildet aus den (vorher definierten) Worten TITEL und RECHNUNG und dem Druckbefehl (."Ende ").

Das neue Wort QUADRAT (das hier schon das ganze Programm darstellt) wird aufgerufen, indem einfach sein Name eingeben wird:

```
QUADRAT
Reihe der Quadratzahlen
1                1
2                4
3                9
4               16
```

```
5          25
6          36
7          49
8          64
9          81
10        100
Ende
OK
```

(Anmerkung: Das OK stammt vom FORTH Interpreter)

Bevor das Wort „QUADRAT“ wie oben definiert werden kann, müssen die Worte TITEL und RECHNUNG definiert werden. Diese Worte können wiederum aus anderen vom Benutzer definierten Worten bestehen, oder (wie hier) direkt aus Worten des FORTH Grundwortschatzes.

Man könnte sagen, die Sprache FORTH besteht nur aus Unterprogrammen, die als Worte bezeichnet werden.
Der Aufbau der Sprache FORTH erlaubt es, den Grundwortschatz sehr „systemnah“ zu definieren. Die im Grundwortschatz der Sprache FORTH vorhandenen WORTE ermöglichen teilweise ganz elementare Funktionen, beispielsweise um den Inhalt einer Speicherzelle in eine andere zu kopieren. Würde die Sprache BASIC nur aus so elementaren Befehlen bestehen, so würde schon ein einfaches Programm sehr lang und sehr aufwendig. In FORTH hingegen lassen sich aus einfachen Worten kompliziertere zusammensetzen, aus denen noch komplexere Worte definiert werden, die ganze Programme darstellen.
Aus diesem Grund benötigt FORTH auch kein Betriebssystem. Die Sprache umfaßt die Befehle, die bei anderen Sprachen das Betriebssystem enthält.

Der Aufbau von FORTH und die im Grundwortschatz vorhandenen Worte erlauben dem Programmierer, mit einer modernen Hochsprache zu arbeiten, und gleichzeitig den Computer in einer Weise zu beeinflussen, wie dies sonst nur in Assembler möglich ist, ohne dessen Nachteile der langwierigen Programmentwicklung in Kauf nehmen zu müssen.

Diese Möglichkeiten der Sprache FORTH werden mit Verantwortung erkauft. FORTH überläßt dem Programmierer sämtliche Kontrollen und Fehlerabfangungen. FORTH kontrolliert nicht, ob ein Befehl zulässig ist oder nicht. Fehlermeldungen wie in BASIC sind nicht vorhanden. Ein falscher Befehl wird

(wenn möglich) ausgeführt, ohne Rücksicht auf Verluste. Ebenso ist die Speicherverwaltung dem Programmierer überlassen.

FORTH ist Compilersprache und Interpreter zugleich. Bei der Definition des Wortes „QUADRAT" wurde der Compiler verwendet. Der Compiler wird durch den Doppelpunkt eingeschaltet und durch den Strichpunkt (am Ende der Zeile) wieder ausgeschaltet; er trägt das neue Wort „QUADRAT" und seine Bedeutung in das Lexikon ein, in dem alle FORTH Worte vorhanden sind. Bei der Ausführung des Wortes „QUADRAT" ausgelöst durch die Eingabe des Wortes selbst, wird der Interpreter aktiv; er sucht das aufgerufene Wort im Lexikon und läßt es ausführen.

Beispiel Interpreter:

```
CR  CR  10  1  DO  I  .  2  SPACES  LOOP  CR

1  2  3  4  5  6  7  8  9  10
OK
```

Die Eingabe in der ersten Zeile (CR CR ...) wird sofort ausgeführt.

Beispiel Compiler:

```
: TEST CR CR 10 1 DO I . 2 SPACES LOOP CR ;
```

Der Compiler wird mit dem Doppelpunkt aufgerufen und definiert das neue Wort, welches TEST heißt und aus den Worten bis zum Strichpunkten besteht. Es wird aufgerufen durch:

```
TEST

1  2  3  4  5  6  7  8  9  10
OK
```

FORTH ist verhältnismäßig klein. Das Betriebsystem und der Grundwortschatz (ungefähr 200 Worte) ist etwa 8 kB groß, also kleiner als die Betriebssysteme DOS 3.3 oder ProDOS. FORTH ist gleichzeitig sehr schnell, je nach Anwendung ca. 25 bis 75% langsamer als Assembler, also um ein Mehrfaches schneller als BASIC. In FORTH geschriebene Programme sind außerdem sehr kompakt.

Hier nun einige Beispiele:

Jedes Drücken der RETURN Taste beantwortet FORTH mit

OK

Der aus BASIC bekannte PRINT Befehl ist in FORTH nur ein Punkt. Hier wird der Punkt in der Version mit dem zu druckenden Text in Anführungszeichen verwendet. Im Gegensatz zu PRINT in BASIC wird nach der Ausführung des Befehls keine neue Zeile angefangen. Dazu dient das Wort CR :

```
 ."Hallo " Hallo OK

CR ."Hallo "
Hallo OK

CR ."Hallo " CR CR
Hallo

OK
```

Wie man am Abstand zwischen CR und dem Punkt sehen kann, werden alle Worte in FORTH durch mindestens ein Freizeichen getrennt:

WORT Abstand WORT Abstand WORT RETURN

Deshalb muß auch nach dem Text „Hallo" ein Freizeichen vor dem abschließenden Anführungszeichen gelassen werden. Sonst wird das abschließende Anführungszeichen nicht als Wort interpretiert, sondern als Teil des Textes aufgefaßt. Als Ausnahme von dieser Regel muß dem Wort ." kein Freizeichen folgen.

4.3.1 Notation

FORTH arbeitet mit der umgekehrten (polnischen) Notation. Verwender der Taschenrechner von Hewlett-Packard kennen sie bereits. Diese umgekehrte Notation sieht die Dateneingabe in umgekehrter Reihenfolge vor. Der BASIC Ausdruck

```
PRINT 3 * 4
```

wird in FORTH geschrieben als

```
4 3 * .
```

und ergibt die Antwort

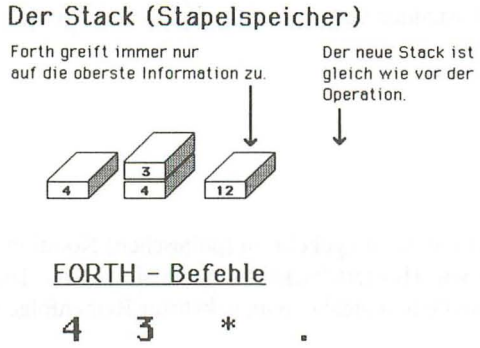
```
12 OK
```

Die umgekehrte Notation ist für den Computer praktischer. Sie erlaubt die einfache Benutzung eines „Stacks" (Stapel). Dies ist wiederum eine Ursache für die hohe Arbeitsgeschwindigkeit von FORTH.

4.3.2 Der Stack

Ein Stapel (Stack) ist eine Organisationsform der Speicherung. Sie funktioniert wie ein Stapel von Spielkarten. Soll eine Information (Zahl) auf dem Stack gespeichert werden, so wird sie „oben drauf" gelegt und der Stapel (Stack) wird höher, genau wie bei einem Stapel Spielkarten. Der Computer hat immer nur Zugriff auf die „oberste" Information, die als letzte eingegeben worden ist; sie entpricht der obersten Karte auf dem Spielkartenstapel. Wird die „oberste" Information weggenommen (und für eine Operation verwendet), kann der Computer auf die vorher zweitoberste Information zugreifen.

Die oben durchgeführte Rechenoperation wird von FORTH mit Hilfe des Stacks erledigt:



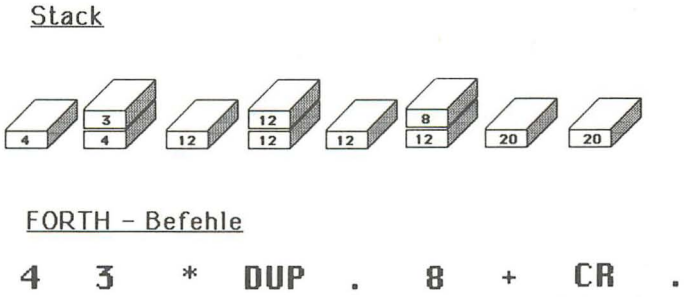
Der FORTH-Interpreter legt die erste Zahl (4) auf den Stack.
Anschließend wird die zweite Zahl (3) auf den Stack gelegt.

Oben auf dem Stack liegt nun die Zahl 3, und darunter die Zahl 4.
Durch den Stern (*) wird eine Multiplikation durchgeführt.
FORTH multipliziert immer die beiden obersten Zahlen auf dem Stack.
Dabei werden die beiden Zahlen vom Stack weggenommen, das Ergebnis (hier 12) auf den Stack gelegt.
Mit dem Punkt (.) wird die oberste Zahl auf dem Stack (auf den Bildschirm) ausgegeben und dabei vom Stack gelöscht.

Die umgekehrte Notation ist also nötig, um die für die Multiplikation notwendigen Zahlen zunächst auf dem Stack zu speichern. Die für den Menschen gebräuchlichere und auch in BASIC verwendete Schreibweise PRINT (3 * 4) ist für den Computer komplizierter, da er mehr Speicheroperationen durchführen muß.

Zur Handhabung des Stacks stellt FORTH einige Befehle zur Verfügung. Soll das Ergebnis der obigen Multiplikation nach der Ausgabe auf den Bildschirm noch weiterverwendet werden, so muß es vor der Ausgabe (mit dem Punkt .) auf dem Stack verdoppelt werden, da es sonst mit der Ausführung des Printbefehls (.) gelöscht wird. Dazu dient der Befehl DUP.
DUP legt die oberste Zahl auf dem Stack noch einmal darauf, sodaß sie zweimal auf dem Stack vorhanden ist.

CR 4 3 * DUP . 8 + CR .
12
20 OK



Das Ergebnis 12 der Rechnung (3 4 *) wird mit DUP zweimal auf den Stack gelegt und einmal mit dem Punkt (.) ausgegeben. Anschließend wird zur noch

vorhandenen Zahl 12 auf dem Stack 8 addiert und dies wieder ausgegeben. Der Stack ist nach dieser Operation wieder wie vorher.
Der Benutzer muß selbst dafür sorgen, daß keine Zahlen auf dem Stack zurückbleiben, sonst wird der Stack schließlich zu groß und „läuft über". Die Höhe des Stacks ist in der Regel auf 255 (größte 8 Bit Zahl) begrenzt.

4.3.3 Progammschleifen und Vergleiche

Die von Basic her bekannten FOR ... NEXT Schleifen entprechen in FORTH den DO ... LOOP's (LOOP = Schleife). Die umgekehrte Notation in FORTH verlangt die Angabe der Grenzen vor dem Wort DO:

```
5 0 DO CR ."Schleife " LOOP
```

```
Schleife
Schleife
Schleife
Schleife
Schleife
Schleife OK
```

In diesem Beispiel ist die erste Zahl (5) die obere Grenze der Zählvariablen, die zweite Zahl (0) ist die untere Grenze. Die Zählvariable der DO ... LOOP's wird mit I aufgerufen.
Der Basic Befehl STEP, der die Schrittbreite festlegt, wird in FORTH durch eine Zahl vor dem Wort +LOOP (statt nur LOOP) definiert:

```
CR 8 0 DO CR I SPACES ."I= " I . 2 +LOOP
```

```
I= 0
  I= 2
    I= 4
      I= 6
        I= 8 OK
```

Hier wird in der Schleife die Zählvariable I angegeben, die jeweils um 2 erhöht wird.
Neben den DO ... LOOPs kennt FORTH die nicht festgelegten Schleifen BEGIN ... UNTIL und BEGIN ... WHILE ... REPEAT.

Mit dem FORTH Wort LEAVE kann ein DO ... LOOP vorzeitig verlassen werden. LEAVE steht häufig in Verbindung mit IF ... THEN. IF ... THEN in FORTH entspricht dem IF ... THEN in Basic, die Syntax ist etwas anders:

BASIC: IF A = 4 THEN PRINT "Alles klar"

Trifft der Vergleich A = 4 zu, dann werden die Befehle zwischen THEN und dem Zeilenende ausgeführt.

FORTH: A 4 = IF ."Alles klar" CR THEN ...

Der Vergleich (A 4 =) findet vor dem IF statt; trifft er zu, dann werden die Befehle zwischen IF und THEN ausgeführt, anschließend die nach dem THEN. Trifft der Vergleich nicht zu, dann werden direkt die Befehle nach THEN ausgeführt.
THEN markiert also das Ende der von IF abhängigen Befehle (in Basic ist es das Zeilenende).

Das FORTH Wort ELSE zwischen IF und THEN trennt Befehle, die bei einem „wahren" Vergleich ausgeführt werden, von denen, die beim „unwahren" Vergleich interpretiert werden:

```
CR CR 4 3 = IF ."4 gleich 3 " ELSE ."4 ungleich 3 " THEN CR
```

4 ungleich 3
OK

4.3.4 Zahlensysteme

Zu den bisher beschriebenen FORTH Worten sind noch Parallelen zu Basic Befehlen vorhanden. Als Beispiel für die Flexibilität der Sprache FORTH im Vergleich zu Basic sollen die Zahlensysteme dargestellt werden:

Basic arbeitet (wie der Mensch) mit dem Dezimalsystem (Zehnersystem). Der Apple //c selbst arbeitet im Dualsystem (Basis 2), der Monitor mit dem Hexadezimalsystem (Basis 16).

FORTH arbeitet mit beliebigen Systemen. Für die Umschaltung vom (gebräuchlichen) Dezimalsystem ins Hexadezimalsystem ist das Wort HEX vorhanden, zurückgeschaltet wird mit DECIMAL.

```
10 CR .
10 OK
10 HEX CR .
A OK
HEX 10 DECIMAL CR .
16 OK
```

FORTH rechnet immer in dem zuletzt definierten Zahlensystem. Um ein beliebiges Zahlensystem zu definieren, dient die Benutzervariable BASE:

```
2 BASE !
```

schaltet FORTH auf das Dualsystem um (das Wort ! speichert den Zahlenwert (2) in der Adresse BASE).

```
36 BASE !
```

schaltet FORTH auf das 36er System um. Das 36er System hat den Vorteil, daß neben den Zahlen 0 bis 9 auch die Buchstaben A bis Z als Zahlen interpretiert werden. Damit können zum Beispiel Strings direkt verglichen werden, weil sie von FORTH als Zahlen interpretiert werden:

```
36 BASE ! ADAM DECIMAL CR .
13654 OK
36 BASE ! EVA DECIMAL CR .
15338 OK
```

4.3.5 Screens und Editor

FORTH verwendet das Konzept des virtuellen Speicherraumes. Disketteninhalte sind für FORTH praktisch nur erweiterter Hauptspeicher. FORTH teilt die Diskette in sogenannte Screens ein, das sind Bereiche von 1 kB (1024 Byte). Diese Screens lassen sich mit dem Befehl LIST (z.B. 88 LIST) gut auf dem Bildschirm darstellen (in 16 Zeilen zu je 64 Zeichen).

Kompliziertere Worte oder Wortgruppen werden nicht direkt in der Doppelpunktform (z.B. : PRODUKT 4 3 * . CR ;) eingegeben, sondern mit Hilfe eines

```
OK
100 LIST
SCR # 100
  0 ( TESTSCREEN        10.11.84     .ST STACK DISPLAY  CUW )
  1
  2 : PST     CR ." ====STACK==== " CR CR ;
  3
  4 : PNR     SP@ - 2 / 4 .R ; ( ? NUMMER GEGENL. ZU I )
  5
  6 : EMPTY? SP@ S0 @ = IF ." Nichts drauf" ( falls Stack leer )
  7           R> DROP 10 SPACES THEN  ;       ( TEXT + DENEST  )
  8
  9 : .ST      PST EMPTY?          ( HEADER UND LEERTEXT )
 10            SP@ 2 - S0 @ 2 - ( VON BOTTOM -> TOS DO )
 11           DO     I PNR        ( . PICK AEQUIVALENT NR )
 12                  I @ 6 .R CR (  UND INHALT, NEXT    )
 13           -2 +LOOP            ( ABWAERTS IN 2 BYTESTEPS )
 14           10 SPACES ;
 15 .ST ;S

OK
```

Editors zunächst in Screens geschrieben. Danach können sie auf Diskette gespeichert werden und der Inhalt der Screens mit dem Befehl LOAD (z.B. 88 LOAD) von der Diskette gelesen, compiliert und ausgeführt werden.

Für Anfänger sind FORTH Programme schwer zu lesen, da der Grundwortschatz im Vergleich zu BASIC groß ist und viele Worte sehr kurz sind. Vom Anwender definierte Worte sollten deshalb nicht zu abstrakt sein und gut mit Kommentaren dokumentiert werden, die in Klammern geschrieben werden.

FORTH ist in vielen verschiedenen Versionen erhältlich. Die Unterschiede zwischen den angebotenen Dialekten betreffen vor allem Apple-spezifisch implementierte Befehle, die im Standard-FORTH nicht vorkommen, besonders bezüglich Grafik. FORTH entwickelt sich relativ rasch, deshalb wurden mehrere Standards definiert, z.B. FORTH 79 und FORTH 83; dadurch ist die Zahl der FORTH Versionen weiter angestiegen.

4.4 Assembler

Der Mikroprozessor 65C02 des Apple //c „versteht" nur Befehle in Zahlenform. Damit ergibt sich für den Benutzer oder Programmierer das Problem, die Maschinenprogramme des Computers lesen zu können. Der Mensch kann nicht a priori unterscheiden, ob eine Zahl des Maschinenprogramms wirklich eine Zahl darstellen soll oder einen Befehl für den Computer. Der Prozessor selbst erkennt diese Unterschiede aus dem Zusammenhang.

20 58 FC A9 CF 8D 00 04 A9 CB 8D 01 04 60

Dies ist ein solches Maschinenprogramm. Alle diese Zeichen sind Zahlen, auch die, die Buchstaben enthalten. Die Verwendung von Buchstaben in Zahlen kommt daher, daß der Computer nicht im Zehnersystem (Dezimalsystem) rechnet wie der Mensch, sondern im 16ersystem (Hexadezimalsystem). Um hexadezimale Zahlen von dezimalen Zahlen zu unterscheiden, werden hexadezimale Zahlen oft mit dem Anhang „hex" versehen oder mit einem Dollarzeichen ($). In dezimalen Zahlen heißt dieses Programm:

32 88 252 169 207 141 0 4 169 203 141 1 4 96

Um hexadezimale Zahlen schreiben zu können, müssen auch die Buchstaben A bis F verwendet werden, weil die Ziffern 0 bis 9 nicht ausreichen:

hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
dez	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

Die größte zweistellige hexadezimale Zahl ist $FF (dezimal 255). In einem Byte kann eine Zahl zwischen $00 und $FF (0 bis 255) gespeichert werden.

4.4.1 Die Eingabe kurzer Maschinenprogramme in den Computer

Um das oben beschriebene Programm in den Computer einzugeben, müssen wird zunächst wissen, an welcher Stelle des RAM-Speichers dieses Programm einmal stehen soll. Im Gegensatz zu einem Basic-Programm, welches selbsttätig vom Computer an die „richtige" Stelle im RAM geschrieben wird, muß bei einem Maschinenprogramm der Speicherplatz (die Adresse) angegeben werden.
Das kurze Maschinenprogramm (14 Byte) schreiben wir ab der Adresse $300 (dez 768) aufwärts. Der Speicherbereich von $300 bis $3CF (768 bis 975) ist normalerweise nicht durch andere Programme belegt, kann also für eigene Programme benutzt werden.

Es gibt zwei Möglichkeiten der Eingabe eines solche Maschinenprogramms:

4.4.1.1 Die Eingabe über ein Basic Programm

```
10 DATA 32, 88, 252, 169, 207, 141, 0, 4, 169, 203, 141, 1, 4, 96
20 FOR I=1 TO 14: READ A: POKE 767 + I,A : NEXT I
```

Zeile 10 des Programm speichert 14 Zahlen innerhalb des Basicprogramms. In Zeile 20 werden die für das Maschinenprogramm gedachten Zahlen mit dem READ Befehl gelesen, und mit POKE (Adresse,Wert) werden sie dezimal an die vorgesehene dezimal angegebene Stelle im RAM geschrieben.

Die richtige Eingabe des Programms wird mit einem Basic Programm geprüft:

```
30 FOR I=768 TO 771: PRINT PEEK ( I ) ; " " ; : NEXT I
```

GOTO 30

32 88 252 169 207 141 0 4 169 203 141 1 4 96

Der Inhalt der Speicherplätze 768 bis 771 wird wieder dezimal angegeben.
Das Maschinenprogramm kann nun gestartet werden mit dem Befehl CALL (dezimale) Startadresse:

CALL 768

Das Programm löscht den Bildschirm und meldet:

OK

Oft liegen Maschinenprogramme allerdings nicht dezimal, sondern (wie in Zeitschriften) hexadezimal vor. Dann empfiehlt sich die direkte Eingabe:

4.4.1.2 Direkte Eingabe von Maschinenprogrammen

Für die direkte Eingabe dient der Monitor. Dieser hat mit einem Bildschirm-Monitor nichts zu tun. Der hier gemeinte Monitor ist ein Maschinenprogramm, welches den direkten Zugriff auf den Speicher erlaubt. Dieser Monitor arbeitet nur mit hexadezimalen Zahlen. Er wird aus Basic aufgerufen durch den Befehl:

CALL -151

*

Statt mit der eckigen Klammer wie Basic meldet sich der Monitor mit dem Stern „*".
Wir geben nun das Maschinenprogramm ein, indem wir zunächst die Speicheradresse des ersten Bytes angeben und, getrennt durch einen Doppelpunkt, die zu speichernden hexadezimalen Werte:

```
300:20 58 FC A9 CF 8D 00 04 A9 CB 8D 01 04 60
```

Nach dem abschießenden Return werden diese Werte in Speicherzelle 300 (hex) und folgende geschrieben; der Monitor meldet sich wieder mit dem Stern. Die richtige Eingabe läßt sich überprüfen mit dem Befehl:

```
300L

0300-  20 58 FC     JSR    $FC58
0303-  A9 CF        LDA    #$CF
0305-  8D 00 04     STA    $0400
0308-  A9 CB        LDA    #$CB
030A-  8D 01 04     STA    $0401
030D-  60           RTS
030E-  00           BRK
030F-  00           BRK
.
.
usw.
```

Der Befehl 300L (L wie List) zeigt von Adresse 300 (hex) an 20 Zeilen Speicherinhalt des RAMs. Dabei wird der Inhalt der Speicherzellen rückübersetzt (disassembled):

```
0300-          20 58 FC          JSR  $FC58

Speicher-      Speicher-         Rückübersetzter Befehl,
adresse des    inhalt der        Assemblerbefehl.
ersten Bytes.  ersten 3 Bytes.
               (hier 300 bis 302)
```

Der Disassembler interpretiert den zu übersetzenden Speicherwert als Befehl. Diese Assemblerbefehle sind zwischen einem und drei Byte lang. Der hier zunächst übersetzte Befehl 20 (JSR) zum Beispiel ist immer drei Byte lang; die beiden folgenden Bytes sind eine Sprungadresse (beim Prozessor des Apple werden die beiden Teile der Adresse in umgekehrter Reihenfolge gespeichert). JSR ist nämlich ein Sprungbefehl (JSR bedeutet Jump to SubRoutine). Das nächstfolgende Byte des Maschinenprogramms ist wieder ein Befehl: A9 bedeutet LDA (LoaD Accumulator, lese einen Wert in den Akkumulator). Der LDA Befehl ist zwei Byte lang, das zweite Byte (hier CF) ist der im Akkumulator zu

speichernde Wert. Der Akkumulator ist ein Teil des Mikroprozessors. In diesem kann eine Zahl gespeichert werden. Der umgekehrte Befehl zu LDA ist der Wert in Speicherzelle 305, der Befehl STA (STore Acumulator, speichere den Wert im Akkumulator in der anschließend angegeben Speicherzelle); STA ist drei Byte lang, die beiden folgenden Bytes sind die Adresse, in der der Wert des Akkumulators gespeichert werden soll.

Damit ist nun klar, wie der Prozessor die Werte in den Speicherzellen als Befehle oder als Zahlen erkennt: Er interpretiert den ersten Wert als Befehl und stellt fest, welche Argumente dazugehören (entweder eine 1-Byte Zahl oder ein 2-Byte Adresse). Einige Befehle haben kein Argument, z.B. der im Programm letzte Befehl RTS (ReTurn from Subroutine). Der gesamte Befehl (mit Argument) wird vom Disassembler in eine Zeile geschrieben und übersetzt. Der Wert der nachfolgenden Speicherzelle wird wieder als Befehl interpretiert.

Wir können das Programm starten mit

300G

Der Befehl 300G (G wie GO) führt das Programm aus, welches an Adresse 300 (hex) steht. Das Programm löscht den Bildschirm und schreibt

OK

Danach meldet sich wieder der Monitor mit dem Stern.
Welche Befehle des Maschinenprogramms führen was aus?

JSR $FC58	ist ein Sprung in ein Unterprogramm, beginnend ab Adresse $FC58, welches den BASIC-Befehl HOME durchführt, also den Bildschirm löscht.
LDA #$CF	Liest den Wert CF in den Akkumulator.
STA $0400	Speichert den Wert CF in Adresse 400. Dies ist die Adresse des 1. Zeichens der 1. Zeile auf dem Bildschirm. Der Wert CF erzeugt dort den Buchstaben O.
LDA #$CB	Liest den Wert CB in den Akkumulator.

STA $0401	Speichert den Wert im Akkumulator in Adresse 401, die Adresse des 2. Zeichens der 1. Zeile auf dem Bildschirm. Dort wir mit CB ein K erzeugt.
RTS	Das Ende des Programms. Es erfolgt ein Rücksprung zum aufrufenden Programm, hier zum Monitor.

Wir können das Programm nachträglich ändern oder ergänzen, indem wir einfach die Inhalte der Speicheradressen ändern. Dies geschieht wieder durch nennen der Speicheradresse, Doppelpunkt und Wert.

```
304:CB

309:CF

300L

0300-  20 58 FC    JSR   $FC58
0303-  A9 CB       LDA   #$CB
0305-  8D 00 04    STA   $0400
0308-  A9 CF       LDA   #$CF
030A-  8D 01 04    STA   $0401
030D-  60          RTS
030E-  00          BRK
.
usw.
```

Die Werte der Speicherzellen 304 und 309 wurden vertauscht, das Programm führt nun nach dem Löschen der Seite etwas anderes aus:

```
300G

KO
```

4.4.1.3 Maschinenprogramme auf Diskette speichern

Um das Programm dauerhaft zu speichern, schreiben wir es auf eine Diskette. Analog zum Befehl SAVE heißt der Befehl für Maschinenprogramme BSAVE (B steht für Binary). Bei BSAVE muß allerdings noch angegeben werden, wo das zu speichernde Maschinenprogramm beginnt (hier 768 dezimal) und wie lang es ist, hier 14 Byte dezimal;

BSAVE OK.BIN, A768, L14

Länge des Programms in Byte (dezimal)

Anfangsadresse des Programms (dezimal)

Filename

Anfangsadresse und Länge des zu speichernden Maschinenprogramms können auch hexadezimal angegeben werden, z.B.:

BSAVE OK.BIN, A$300, L$0E.

Unter dem Betriebssystem ProDOS ist statt der Längenangabe auch die Endadresse zulässig; entweder wieder dezimal oder wie hier hexadezimal (mit $ Zeichen):

BSAVE OK.BIN, A$300, E$30E

Mit BLOAD Filename kann das Programm wieder von der Diskette eingelesen werden. Es wird dann an der alten Adresse gespeichert. Analog dazu wird mit BRUN Filename das Maschinenprogramm von der Diskette eingelesen und sofort gestartet. Der Befehl BRUN allein ist nicht zulässig. Soll ein schon im Speicher vorhandenes Maschinenprogramm gestartet werden, so geschieht das von BASIC aus mit dem Befehl CALL Startadresse (dezimal), für das obige Beispiel CALL 768.
Für ein von der Diskette in den Hauptspeicher eingelesenes Maschinenprogramm kann unter DOS 3.3 mit

PRINT PEEK(43635)*256 + PEEK(43634) (Startadresse)

PRINT PEEK(43617)*256 + PEEK(43616) (Programmlänge)

die Startadresse und die Programmlänge festgestellt werden.

4.4.2 Assemblerprogramme

Bisher haben wir nur reine Maschinenprogramme eingeben. Diese Methode ist etwas aufwendig und führt zu Fehlern. Besser und einfacher ist die Eingabe der direkten Befehle, z.B. JSR $FC58. Diese Befehle, auch Assemblerbefehle oder Mnemonics genannt, werden dann von einem Programm in Maschinenbefehle übersetzt, auch assembliert.

In allgemeinen Handbüchern über Apple Computer ist oft vom Miniassembler die Rede. Dieser Miniassembler ist ein Programm, welches Assemblerbefehle in Maschinenprogramme übersetzt.

4.4.2.1 Der Miniassembler

Der Miniassembler ist ein ganz einfacher Assembler ohne Komfort, der die eingegebenen Befehle sofort übersetzt. Er ist nicht für die neuen Befehle der 65C02 CPU ausgelegt. Bei der Eingabe von Maschinenprogrammen aus Zeitschriften oder Büchern (die sich in der Regel mit dem 6502 Befehlssatz begnügen), oder bei der Entwicklung eigener einfacher Maschinenprogramme ist er trotzdem eine große Hilfe.

Auf dem Apple //c ist der Miniassembler nicht vorhanden. Der Miniassembler kann trotzdem aktiviert werden:
Dazu wird mit einer DOS 3.3 Master Diskette gebootet. Diese Diskette muß den Integer Basic-Interpreter enthalten. Normalerweise wird dies beim Booten durch den Hinweis

LOADING INTEGER BASIC INTO LANGUAGE CARD

(oder ähnlich) bestätigt. Anschließend wird mit dem Befehl

INT

der Integer Basic-Interpreter aktiviert (Der Umkehrbefehl von Integer Basic nach Applesoft heißt übrigens FP). Danach wird der Miniassembler aufgerufen:

CALL -2458

Der Miniassembler meldet sich mit dem Ausrufezeichen und einem Ton:

!

Die Befehlseingabe geschieht ähnlich wie im Monitor (das Ausrufezeichen stammt vom Miniassembler, es wird nicht eingegeben):

```
!0300:JSR $FC58
```

Zunächst wird die Speicheradresse angegeben (immer hexadezimal, aber ohne $ Zeichen, da der Miniassembler nur hexadezimale Adressen verarbeitet), anschließend durch Doppelpunkt getrennt der Befehl, dabei Adressen wahlweise mit oder ohne $ Zeichen beginnend. Der Miniassembler übersetzt das sofort und meldet:

```
0300-       20 58 FC       JSR   $FC58
Adresse     hexadezimale   Rückübersetzter
des 1.      Inhalte        Assemblerbefehl
Bytes
```

Bei weiteren Befehlen, die im Speicher direkt anschließend abgelegt werden sollen, ist die Adressangabe nicht mehr notwendig. Sie wird vom Miniassembler selbst berechnet:

```
! LDA #$CB

0303- A9 CB    LDA #$CB
! STA $0400

0305- 8D 00 04 STA $0400
```

usw.

Die Angaben hinter den Ausrufezeichen sind die Eingaben des Benutzers, den Rest generiert der Miniassembler.

Analog wie beim Monitor geschieht auch die Darstellung von Speicherinhalten, hier allerdings mit vorangestelltem $-Zeichen:

```
!$305

0305- 8D
```

Der Miniassembler wiederholt die Adresse und den zugehörigen Inhalt. Der List Befehl (L) stellt in zwanzig Zeilen die Speicherinhalte und die entsprechenden Befehle ab der angegebenen Adresse dar:

```
!$300L

0300- 20 58 FC JSR $FC58
0303- A9 CB    LDA #$CB
0305- 8D 00 04 STA $0400
...
```

usw.

Beim Miniassembler ist immer das $-Zeichen vor den hexadezimalen Zahlen anzugeben, wenn Monitorbefehle ausgeführt werden sollen, ansonsten nicht!

Mit Control-C oder $3D0G erreicht man aus dem Miniassembler wieder den Basic Interpreter.

4.4.2.2 Andere Assembler

Will man eigene Assemblerprogramme (in größerem Stil) entwickeln, so muß zunächst ein Assembler von der Diskette eingelesen werden; solche Assembler sind im Handel erhältlich.
Achtung! Viele Assembler sind für den 6502 Prozessor geschrieben und verstehen nicht alle Befehle der 65C02 CPU.
Die Assembler gestatten die Verwendung symbolischer Adressen und die Eingaben von Kommentaren, um die Lesbarkeit eines Assemblerprogramms zu verbessern:

```
:ASM*, R300.OBJ0


0000:                 1 ***** Messung Paddleeingang /
0000:                 2 ***** Messwerte speichern /
0000:                 3 ***** Startadresse in $06,$07 /
0000:                 4 ***** Endadresse in $08,$09 /
0000:                 5 *****
----- NEXT OBJECT FILE NAME IS R300.OBJ0
0300:        0300     6          ORG   $300
0300:A2 00            7 START    LDX   £$00
0302:C9 01            8          CMP   £$01
0304:F0 2E   0334     9          BEQ   LAST
0306:AD 70 C0        10          LDA   $C070          ;Paddles Trigger
0309:A0 00           11          LDY   £$00
030B:BD 64 C0        12 LOOP     LDA   $C064,X        ;Messloop
030E:10 07   0317    13          BPL   STORE
0310:C8              14          INY
0311:D0 F8   030B    15          BNE   LOOP           ;Ende Messloop
0313:88              16          DEY
0314:4C 17 03        17          JMP   STORE          ;Messwert abspeichern
0317:98              18 STORE    TYA
0318:A2 00           19          LDX   £$00
031A:81 06           20          STA   ($06,X)
031C:18              21          CLC
031D:E6 06           22          INC   $06            ;Speicheradresse
031F:A5 06           23          LDA   $06            ;aendern
0321:C9 FF           24          CMP   £$FF
0323:D0 DB   0300    25          BNE   START
0325:E6 07           26          INC   $07
0327:A5 06           27          LDA   $06
0329:C5 08           28          CMP   $08            ;Messung fertig ?
032B:D0 D3   0300    29          BNE   START          ;Nein -  Continue
032D:A5 07           30          LDA   $07
032F:C5 09           31          CMP   $09
0331:D0 CD   0300    32          BNE   START
0333:60              33 END      RTS                   ;Ende der Messung
```

```
0334:E0 01          34 LAST     CPX   £$1
0336:6A             35          ROR
0337:A8             36          TAY
0338:B9 76 05       37          LDA   $0576,Y
033B:F0 06   0343   38          BEQ   RET
033D:A9 FF          39          LDA   £$FF
033F:19 76 04       40          ORA   $0476,Y
0342:A8             41          TAY
0343:60             42 RET      RTS

?0333 END            0334 LAST          030B LOOP          0343 RET

 0300 START          0317 STORE
** SUCCESSFUL ASSEMBLY := NO ERRORS
** ASSEMBLER CREATED ON 21-MAY-83 REL-07
** TOTAL LINES ASSEMBLED     42
** FREE SPACE PAGE COUNT    96
```

Da die Programmierung in Assembler gegenüber BASIC sehr aufwendig ist, beschränkt man sich häufig auf die Formulierung der zeitkritischen Sequenzen in Assembler, den Rest wird in Basic programmiert (oder einer anderen Sprache). Kleine Assemblerprogramme können relativ übersichtlich gehalten werden, und nach der Formulierung in Assembler-Mnemonics mit Hilfe einer Tabelle (siehe Anhang) von Hand oder mit dem Miniassembler übersetzt werden.

4.4.2.3 Assemblersyntax

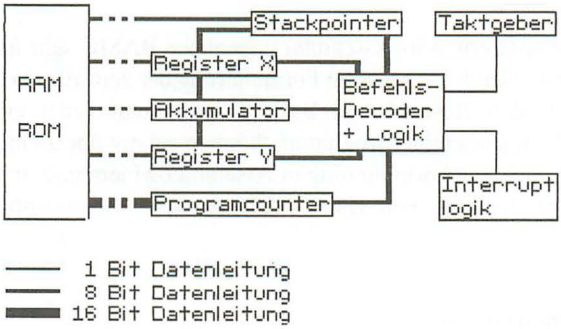
In dem Programmbeispiel haben wir einige einfache Assemblerbefehle kennengelernt. Im Folgenden werden die wichtigen 65C02 Assemblerbefehle erklärt. Zunächst aber zur Funktionsweise des Mikroprozessors:

Der Mikroprozessor des Apple //c besteht aus dem Befehlsdecoder mit Logikeinheit und mehreren internen „Speicherplätzen", sogenannten Registern, die er zur Ausführung von Operationen benötigt. Die wichtigsten drei Register sind:
Akkumulator
X-Register
Y-Register

Jedes dieser Register kann eine 8 Bit Zahl (zwischen 0 und 255) enthalten. Weiter sind an den Befehlsdecoder und die Logikeinheit (ALU) ein Taktgeber angeschlossen, der die Arbeitsfrequenz des Prozessors bestimmt (ca. 1 Megahertz), sowie Interruptlogik, Prozessorstatusregister, Stackpointer und Programcounter. Der Stackpointer enthält die „Höhe" des Stacks (Stapel), damit

den Ort der aktuellen Information des Stacks. Ein Stack ist ein Stapel mit Informationen, etwa vergleichbar mit einem Stapel Spielkarten.
Der Prozessorstatus (oder Statusregister) enthält Informationen über Vergleichsergebnisse, das aktuelle Vorzeichen usw.
Der Programcounter enthält die (16 Bit) Adresse des momentan zu verarbeitenden Bytes im Speicher. Der Programcounter wird bei jedem Schritt um eins erhöht, um den Zugriff auf das nächste Byte im Programm zu ermöglichen. Bei Sprüngen im Programm (JMP oder JSR) wird der Programcounter entsprechend geändert.

Funktionsschema der 65C02 CPU



Der Mikroprozessor 65C02 kann bis auf wenige Ausnahme nur Operationen an den Inhalten seiner Register ausführen. Deshalb beziehen sich die meisten Befehle auf den Datenaustausch mit den Registern, besonders auf den Akkumulator.

Die Assemblerbefehle lassen sich grob einteilen in die Gruppen:

1. Speicher-, Lese- und Transferbefehle
2. Arithmetische Operationen
3. Stackoperationen
4. Logische Operationen
5. Vergleichsoperationen
6. Verzweigungsbefehle
7. Testbefehle
8. Interruptbefehle

Zu jeder dieser Gruppen gehören ein oder mehrere Befehle, von denen die meisten wiederum in verschiedenen Versionen (unterschiedliche Adressie-

rungsarten) möglich sind. Anhand des bereits aus dem Beispielprogramm bekannten Befehls LDA (Liest einen Wert in den Akkumulator) sollen zwei dieser Versionen erläutert werden:

4.4.2.4 Lade und Speicherbefehle

LDA ist die Abkürzung für LoaD Akkumulator. Mit diesem Befehl wird ein Wert in den Akkumulator gelesen.

LDA $200

Diese Schreibweise liest den in Speicherzelle 200 hex. enthaltenen Wert in den Akkumulator ein. Die Übersetzung in Maschinensprache heißt:

AD 00 02 LDA $200

Wert in Speicherstelle 200 hex ----> Akkumulator

Eine andere Version des Befehls LDA liest statt dem Inhalt einer Speicherzelle einen bestimmten Wert direkt in den Akkumulator:

LDA #$FF liest den Wert FF (255 dezimal) in den Akkumulator.

Zahl FF ---> Akkumulator

Übersetzt in Maschinensprache heißt dieser Befehl:

A9 FF LDA #$FF

Außer diesen beiden wichtigen LDA-Versionen AD und A9 existieren noch sechs weitere.

Insgesamt erlaubt der 65C02 Prozessor 15 verschiedene Arten der Adressierung. Die verschiedenen Adressierungsarten sind im Anhang erläutert.

Genau analog wie LDA arbeiten die Befehle LDX und LDY, welche einen Wert in das Register X bzw. in das Register Y einlesen. Von LDX und LDY existieren „nur“ je fünf Versionen.

LDX $200	liest den Wert in Speicherzelle 200 hex in das X Register

Ein im Akkumulator vorhandener Wert kann mit dem Befehl STA (STore Akkumulator) gespeichert werden:

STA $201	Akkumulator ---> Speicherzelle 201

speichert den Wert im Akkumulator in Speicherzelle 201 (hex). STA ist in sieben Varianten möglich. Die Parallelbefehle STX und STY, die die Inhalte der Register X bzw. Y abspeichern, existieren in je drei Versionen.

Einige Befehle dienen dem Datenaustausch im Prozessor selbst. Es sind die Befehle TAX, TAY, TXA und TYA; sie transferieren den Inhalt der Register X und Y zum Akkkumulator bzw. zurück. Ein direkter Austausch zwischen dem X und dem Y Register ist nicht vorgesehen, dazu muß der Akkumulator als Zwischenspeicher dienen.

TXA	Inhalt im X Register	---->	Akkumulator
TYA	Inhalt im Y Register	---->	Akkumulator
TAX	Inhalt im Akkumulator	---->	X Register
TAY	Inhalt im Akkumulator	---->	Y Register

4.4.2.5 Befehle für arithmetische Operationen

Mit den Befehlen INA (bzw. INX, INY) wird der Wert im Akkumulator (bzw. Register X, Y) um eins erhöht. Die Befehle DEA, DEX und DEY vermindern der Wert im Akkumulator bzw. in den X, Y Registern um eins. Diese Befehle werden vor allem bei der Programmierung von Schleifen (analog zu den FOR ... NEXT Schleifen in Basic) benötigt.

INA	erhöht den Wert im Akkumulator um eins. (INA hat kein Argument)
DEA	vermindert den Wert im Akkumulator um eins. (DEA hat kein Argument)

ADC ist ein Additionsbefehl; er addiert den Inhalt einer Speicherzelle zum Inhalt des Akkumulators und legt das Ergebnis im Akkumulator ab.

ADC addiert Speicherzelleninhalt zum Akkumulator

Dabei ist es möglich, daß das Ergebnis größer wird als 255. Um diese Zahl zu speichern, sind mehr als 8 Bit notwendig. Im Statusregister ist deshalb ein zusätzliches Bit dafür reserviert, das Übertragsbit. Dieses Bit (sozusagen das 9. Bit) wird gleich eins gesetzt, wenn das Ergebnis der Addition größer als 255 ist. Beispiel:

Bitnummer:	(9) 8	7	6	5	4	3	2	1	hex	dez
Addition von Speicherzelle :	1	0	1	0	0	0	1	1	A3	163
und Akkumulator :	0	1	0	0	1	1	1	0	4E	78
Ergebnis :	(0) 1	1	1	1	0	0	0	1	F1	241

Bitnummer:	(9) 8	7	6	5	4	3	2	1	hex	dez
Addition von Speicherzelle :	1	1	1	0	0	0	1	1	E3	227
und Akkumulator :	0	1	0	0	1	1	1	0	4E	78
Ergebnis :	(1) 0	0	1	1	0	0	0	1	131	305

Im zweiten Beispiel ist das Ergebnis größer als 255, deshalb wird das Übertragsbit gleich 1 gesetzt. Um das richtige Ergebnis zu erhalten, muß gewährleistet sein, daß das Übertragsbit vorher gleich 0 war; dazu dient der Befehl CLC (Clear Carry bit). Eine Addition sieht beispielsweise so aus:

```
18          CLC          löscht das Carry bit
6D 44 03    ADC $0344    addiert Speicherzelle $344
                         zum Akkumulator.
```

ADC ist in neun verschiedenen Adressierungsarten möglich.

Analog zu ADC (Addition) funktioniert der Befehl SBC; er subtrahiert den Inhalt einer Speicherzelle vom Akkumulator. Damit das Ergebnis nicht kleiner als 0 wird, muß das Übertragsbit vorher auf 1 gesetzt werden. Anschließend an die Subtraktion wird geprüft, ob das Übertragsbit gleich 0 ist; in diesem Fall wäre das Ergebnis kleiner als 0 geworden.

```
38          SEC          setzt das Carry bit auf 1
ED 44 03    SBC $0344    subtrahiert Speicherzelle
                         $344 vom Akkumulator
```

Die Befehle ASL und LSR verschieben die 8 Bits eines Bytes um je eine Stelle nach links (ASL), also Bit 0 wird Bit 1, Bit 1 wird Bit 2, etc. Bit 7 wird zum Carry Bit, bzw. nach rechts (LSR). Diese Operationen entsprechen einer Multiplikation (ASL) mit 2, bzw. einer Division (LSR) durch 2.

4.4.2.6 Stackoperationen

Der Befehl PHA (Push Accumulator onto stack) transferiert (schiebt) den Inhalt des Akkumulators zum (auf den) Stack. Die Information liegt dann „oben" auf dem Stack.

PHA	„schiebt" den Akkumulatorinhalt auf den Stack
PLA	„zieht" den obersten Stackinhalt auf den Akkumulator

Der Befehl PLA (Pull Accumulator from stack) überträgt den obersten Wert auf dem Stack in den Akkumulator und reduziert die Höhe des Stacks um eins (Das Wesen von Stackoperationen ist im Abschnitt FORTH beschrieben).

4.4.2.7 Logische Operationen

Logische Operationen sind AND, EOR und ORA. Alle drei verknüpfen den Inhalt des Akkumulators mit dem Inhalt einer Speicherzelle. Die Verknüpfungsarten sind:

AND	logisches UND
EOR	logisches exklusives ODER
ORA	logisches ODER

Das Ergebnis der Verknüpfungen steht im Akkumulator.

Wahrheitstabellen der logischen Operationen

Ergebnisse (im Akku) nach Operation:

Akkubit	Speicherbit	AND	EOR	ORA
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	0	0

4.4.2.8 Vergleichsoperationen

Vergleichsbefehle sind im Prinzip logische Befehle. Sie vergleichen (subtrahieren) den Inhalt einer Speicherzelle mit dem (vom) Akkumulator, X oder Y Register. Je nach Ergebnis werden im Statusregister Bits gesetzt.

CMP	Compare	vergleicht Akku mit Speicherzelle, setzt Z=1, wenn gleich, setzt N=1, wenn negativ.
CPX	Compare with X	vergleicht X-Register mit Speicherzelle, setzt Z und N wie CMP.
CPY	Compare with Y	vergleicht Y-Register mit Speicherzelle, setzt Z und N wie CMP.

4.4.2.9 Verzweigungsbefehle

Die einfachsten Verzweigungsbefehle sind JMP (entpricht GOTO in Basic) und JSR (entpricht GOSUB in Basic). RTS entpricht in Basic dem RETURN.

JMP $0300	führt Sprung zu Adresse $300 aus
JSR $0300	führt Sprung zu Adresse $300 aus, schiebt Rücksprungadresse auf den Stack

RTS springt zur Rücksprungadresse

Daneben gibt es eine Anzahl von Verzweigungsbefehlen, die von Werten im Statusregister abhängig sind. Die Verzweigungen sind relativ; es ist also nur ein Sprung um einen bestimmten Betrag (0 bis 127) nach vorne oder nach hinten in der Speichernumerierung möglich.

Befehl	Bezeichnung	verzweigt, wenn
BCC	Branch if Carry clear	Carrybit C = 0
BCS	Branch if Carry set	Carrybit C = 1
BEQ	Branch if Equal	Ergebnisbit Z = 1
BNE	Branch not Equal	Ergebnisbit Z = 0
BPL	Branch if Plus	Vorzeichenbit N = 0
BMI	Branch if Minus	Vorzeichenbit N = 1

Die Bits C, Z und N werden meist durch Vergleichsoperationen gesetzt.

Der 65C02 Prozessor des Apple//c besitzt (neu) den Befehl BRA. BRA führt einen unbedingten relativen Sprung aus.

Ein relativer Sprung wird immer von der Adresse nach dem Sprungbefehl berechnet, weil der Programcounter bereits weitergezählt hat.

4.4.2.10 Interruptbefehl

Der Interruptbefehl ist BRK (Break). Ein Interruptbefehl unterbricht den normalen Ablauf eines Programms, speichert den momentanen „Zustand" des Prozessors ab und führt einen Sprung zu der Adresse aus, die in $FFFE und $FFFF steht (normalerweise $C803, die Interrupt Handling Routine).
Außer durch BRK kann ein Interrupt durch den IRQ-Anschluß an den Prozessor ausgelöst werden (IRQ= Interrupt Request). Dazu muß vorher ein CLI (Clear Interrupt Disable Flag) ausgeführt worden sein. Durch SEI (Set Interrupt Disable Flag) wird ein möglicher IRQ Interrupt verhindert.
Der Apple //c besitzt im Gegensatz zu den älteren Geräteversionen //plus und //e eine ausgefeilte Interrupt-Steuerung, da sowohl die Maus als auch die seriellen Schnittstellen mit Interrupts arbeiten. Appendix E des Reference Manuals enthält eine ausführliche Beschreibung.

4.5 Monitorbefehle

Der Monitor wird aus dem Basic Interpreter mit CALL -151 aufgerufen und meldet sich mit dem Stern:

*

---------- **WARNUNG** ----------

Die unüberlegte Arbeit mit dem Monitor kann Systemprogramme ändern und einen Neustart (Reboot) notwendig machen. Besonders die Speicherzellen $C000 bis $C0FF (die spezielle Hardwareinformationen enthalten) rufen durch Lesen oder Schreiben einen unkontrollierten Systemzustand hervor.

Der Monitor ist ein Hilfsprogramm, das die direkte Arbeit mit Speicherzellen und Maschinenprogrammen erlaubt. Der Monitor besitzt dafür einige Befehle:

4.5.1 Speicherinhalte listen und disassemblieren

Oben wurde bereits der Listbefehl beschrieben,

*300L

disassembliert 20 Zeilen (einen Bildschirm) von der angegebenen hexadezimalen Adresse (hier 300) aufwärts:

```
0300-  20 58 FC     JSR    $FC58
0303-  A9 CF        LDA    #$CF
0305-  8D 00 04     STA    $0400
```

usw.

L disassembliert dann weitere 20 Zeilen. Führende Nullen können weggelassen werden.

4.5.2 Speicherinhalte abfragen

Eine einzelne Speicherzelle kann durch Angabe ihrer Adresse abgefragt werden; die Eingabe

*300

liefert die Adresse mit Inhalt

```
0300- 20
```

Durch die Eingabe eines Punktes und einer weiteren Adresse werden mehrere Speicherinhalte gleichzeitig ausgegeben:

*.30A

```
0301- 58 FC A9 CF 8D 00 04
0308- A9 CF 8D 01
```

(Start bei Adresse $301, weil die letzte verwendete Adresse $300 war).

*CDBE.CDCB

```
CDBE- 38 90
CDC0- 18 2C FB 04 10 54 08 20
CDC8- 1B CE 28 80 08
```

4.5.3 Speicheradressen ändern

Speicheradressen werden durch Angabe der Adresse und des zu speichernden Wertes, getrennt durch den Doppelpunkt, geändert:

*300:20

Mehrere Speicherzellen hintereinander können durch Angabe der ersten Adresse und der zugehörigen Werte, voneinander durch Abstand (Leerzeichen) getrennt, verändert werden:

*300: 20 3A FF

*300.302

```
300- 20 3A FF
```

4.5.4 Speicherinhalte kopieren

Der Kopierbefehl heißt M (M für Move):

Ziel < Start.Ende M

Wobei „Ziel“ die untere Grenze des Zielbereichs angibt, „Start“ die untere Grenze des Originalbereichs und „Ende“ die Obergrenze des Originalbereichs. Die Kopie erfolgt byteweise, beginnend mit der niedrigsten Originaladresse. Beispiel:

*380 < 300.320M

kopiert den Inhalt der Speicherzellen $300 bis $320 zu Speicherzelle $380 und folgende (bis $3A0).
Mit dem Kopierbefehl M lassen sich Speicherbereiche mit Binärmustern füllen:

*300:A0
*301 < 300.320M

setzt Speicherzelle $300 auf $A0 und kopiert (byteweise) den Bereich $300 bis $320 zum Bereich $301 ff. (bis $321). Zunächst wird $300 nach $301 kopiert, $301 enthält jetzt auch eine $A0. Anschließend wird $301 nach $302 kopiert usw., bis der gesamte Bereich $A0 enthält:
Kontrolle:

*300.320

```
0300- A0 A0 A0 A0 A0 A0 A0 A0
0308- A0 A0 A0 A0 A0 A0 A0 A0
0310- A0 A0 A0 A0 A0 A0 A0 A0
0318- A0 A0 A0 A0 A0 A0 A0 A0
0320- A0
```

oder:

*301:00 (schreibt $00 in $301)

*302 < 300.320M (kopiert $300 bis $320 nach $302ff.)

*300.320 (überprüft das Resultat)

```
0300- A0 00 A0 00 A0 00 A0 00
0308- A0 00 A0 00 A0 00 A0 00
0310- A0 00 A0 00 A0 00 A0 00
0318- A0 00 A0 00 A0 00 A0 00
0320- A0
```

Analog zu Move (M) lassen sich mit Speicherbereiche vergleichen:
Syntax:

Ziel < Start.Ende V

Beispiel:

*310 < 300.30FV

Im Beispiel werden byteweise die Inhalte der Speicherbereiche $300 bis $30F und $310 ff. verglichen. Stimmen zwei Inhalte nicht überein (Beispiel: bei vorhergehendem 311:01), so wird die Adresse im (in der Anweisung rechten) Bereich angegeben und die dazugehörigen Werte:

0301-01 (00)

4.5.5 Sprünge im Monitor

Im Monitor können Maschinenprogramme gestartet werden durch die Anweisung:

StartadresseG (G steht für GO). Beispiel:

*FB4EG

startet ein Maschinenprogramm ab Adresse $FB4E. Mit der Anweisung

*3D0G

wird der Basic Interpreter gestartet (eine Möglichkeit, den Monitor wieder zu verlassen).

Ähnlich wie der GO Befehl wirkt CONTROL-Y. Dieser Befehl führt immer einen Sprung nach $3F8 aus (startet dort ein vom Benutzer definierbares Maschinenprogramm).

4.5.6 Ausgabe über Drucker

Die Monitorinformationen können über Drucker ausgegeben werden. Der Drucker wird durch

1 CONTROL-P (RETURN)

eingeschaltet. Dabei wird der serielle Ausgang 1 angesprochen, an dem normalerweise der Drucker hängt.
Mit der Eingabe

0 CONTROL-P

wird wieder der Bildschirm aktiviert (dies ist auch mit Esc-CONTROL-Q möglich). Mit

Slotnummer CONTROL-K

wird der Eingang auf dem Slot Slotnummer gelegt. Die Befehle CONTROL-K und CONTROL-P entsprechen den Basic Befehlen IN# und PR#.

4.5.7 Registerinhalte des Prozessors ändern

Mit CONTROL-E werden die Inhalte der Register

Akkumulator	A
X Register	X
Y Register	Y
Statusregister	P
Stackpointer	S

angegeben:

*CONTROL-E

```
A=A0 X=FD Y=44 P=B0 S=F4
```

Die Registerinhalte können (nach der Ausgabe durch CONTROL-E) durch Eingabe nach einem Doppelpunkt in der Reihenfolge ihrer Darstellung (durch CONTROL-E) geändert werden:

*:A1 FE 00 00

Erneute Kontrolle durch

*CONTROL-E

```
A=A1 X=FE Y=00 P=00 S=F4
```

zeigt die geänderten Registerinhalte; ungeänderte Registerinhalte (am Schluß) brauchen beim Ändern nicht eingeben zu werden (hier S).

4.5.8 Videoeffekte

Die Eingabe I schaltet auf inverse Darstellung der folgenden Darstellungen. Die Eingabe N schaltet wieder auf normale Darstellung zurück.

4.5.9 Hexadezimale Arithmetik

Der Monitor kann Additionen und Subtraktionen innerhalb des 8 Bit Bereichs (0 bis 255) ausführen:

*10+10
=20

*5A-C
=4E

*7-9
=FE

Geht das Ergebnis über den 8-Bit Bereich hinaus, so werden falsche Resultate angegeben.

4.5.10 Verlassen des Monitors

Normalerweise erfolgt aus dem Monitor der Rücksprung in den Basic-Interpreter; dies ist auf mehrere Arten möglich:

*3D0G springt nach 3D0, schaltet den Basic-Interpreter ein.

*CONTROL-C oder

CONTROL-RESET

schalten den Basic-Interpreter ein. Ein vorhandenes Basic-Programm oder Daten werden nicht zerstört.

*CONTROL-B springt auch zum Basic-Interpreter, jedoch werden dabei Programme und Daten gelöscht.

Kapitel 5

Anschluß externer Geräte

5.1 Die Slots/Ports des Apple //c

Befehle: **PR#, IN#, (Cntrl-D)**

Der Apple //c ist mit zwei seriellen Anschlüssen ausgerüstet, zudem mit einem Anschluß für eine Maus, Joystick o.ä. Um den Zugriff auf diese „Schnittstellen" zu erklären, muß das Hardwarekonzept des Computers kurz dargestellt werden.

Die Betriebssysteme DOS 3.3 und ProDOS bezeichnen die verschiedenen Teile des Computers als Slots (Steckplätze). Wie schon erwähnt, sind diese Slots nur im Parallelmodell Apple //e und den Vorgängern physisch vorhanden, nicht im Apple //c. Diese Slots sind von 0 bis 7 durchnumeriert; sie haben den Charakter von Ein- oder Ausgängen, zu denen der Prozessor Kontakt hat, von denen er Informationen erwartet oder an die er Daten weitergibt. Aus diesem Grund wird für die seriellen Anschlüsse beim Apple //c auch der Begriff Ports verwendet.

Der Prozessor kann wegen der Struktur der Betriebssysteme immer nur von jeweils einem dieser Slots Daten empfangen.

Der eigentliche ursprüngliche Slot hat die Nummer 0. Als Eingang dient dann die eingebaute Tastatur und als Ausgang der Monitor- oder der Fernsehanschluß an der Computerrückseite in 40 Zeichendarstellung. Dies ist der normale Zustand des Computers nach dem Einschalten.

Der Computer kann auf andere Ein- und Ausgänge umgeschaltet werden durch Eingaben der Tastatur oder Anweisungen aus einem Programm.

Die Slots sind beim Apple //c im Gegensatz zu anderen Apple // Computern fest vergeben:

Slot Nr.	Bedeutung als Eingang	Bedeutung als Ausgang
0	Tastatur	40 Zeichen Darstellung über Fernseh- oder Monitoranschluß
1	Serieller Eingang zu verschiedenen Zwecken	Serieller Ausgang zu verschiedenen Zwecken, vor allem für Drucker
2	Serieller Eingang zu verschiedenen Zwecken (z.B. Modem)	Serieller Ausgang zu verschiedenen Zwecken, vor allem für Modem
3	Tastatur	80 Zeichen Darstellung über Fernseh- oder Monitoranschluß
4	Anschluß für Maus etc.	Ausgang an Mausanschluß zu verschiedenen Anwendungen
5	ohne Bedeutung	ohne Bedeutung
6	Eingang von eingebautem Diskettenlaufwerk	Ausgang zu eingebautem Diskettenlaufwerk
7	Eingang für weiteres externes Diskettenlaufwerk	Ausgang für weiteres externes Diskettenlaufwerk

Das eingebaute Diskettenlaufwerk und die Hardware für die 80 Zeichendarstellung sind zwar in das Gehäuse des Apple //c integriert, der Rechner selbst bzw. das Betriebssystem behandelt sie aber als externe Geräte, weil das Betriebssystem vom Apple //e stammt, wo Diskettenstation und 80 Zeichenkarte tatsächlich erst an den Computer angeschlossen werden. Der Computer sieht sich selbst nur als Slot 0. Physisch ist das nur ein Teil der Platine im Apple //c, alles andere ist für die Maschine Umwelt, mit der sie nur über die Slots 1 bis 7 kommuniziert.

Die jeweiligen Slots können durch die Befehle IN# (Eingabe) und PR# (Ausgabe) unabhängig voneinander angesprochen werden:

PR#3

schaltet die Ausgabe auf einen Baustein (im imaginären Slot 3), der die 80 Zeichendarstellung besorgt und über den Monitor- oder Fernsehausgang ausgibt.

IN#2

macht die Tastatur (mit Ausnahme von reset) unwirksam, der Prozessor erwartet Eingaben über den 2. seriellen Eingang.

PR#1

schaltet die Ausgabe über den seriellen Ausgang 1 z.B. zu einem Drucker.
Nicht alle IN# oder PR# haben diese Input- oder Outputwirkung. Das Verhalten des Systems Computer/Anschlußgerät hängt dabei im wesentlichen vom angeschlossenen Gerät ab. Beispielsweise wird bei IN#6 oder PR#6 vom eingebauten Diskettenlaufwerk gebootet, also alles reinitialisiert und neu angefangen. Im Computer gespeicherte Programme oder Daten gehen dabei verloren.

Wird durch PR# oder IN# ein blinder Ein- oder Ausgangskanal angesprochen, z.B. durch PR#1 ohne angeschlossenen Drucker, so kann das System hängenbleiben. In diesem Fall erwartet der Computer vom angesprochenen Kanal eine Eingabe, die er nicht erhält. Solche Fälle lassen sich durch CONTROL-RESET lösen. Der Computer schaltet dann Ein- und Ausgabe wieder auf Slot 0.
Sollen die Befehle PR# und IN# innerhalb von Programmen verwendet werden, so muß unmittelbar vor dem Befehl Control-D ausgegeben werden, da sich PR# und IN# an das Betriebsystem wenden und nicht an den APPLESOFT-Interpreter:

```
5 D$ = CHR$(4) : REM - Control-D
10 INPUT"Bitte Datum und Zeit:" ; A$
20 PRINT D$ ; "PR#1"
30 PRINT A$ : PRINT
40 PRINT D$ ; "LIST"
50 PRINT D$ ; "PR#3"
60 END
```

Das Programm druckt sein eigenes Listing mit (immer empfehlenswerter) Datumsangabe auf einen an Slot 1 angeschlossenen Drucker. Anschließend wird auf den Bildschirm mit 80 Zeichendarstellung geschaltet.

Bemerkung: Solche kurzen Programme können mit hohen Zeilennummern versehen (z.B. 30000 bis 30060) und bei Bedarf mit GOTO 30000 aufgerufen werden. Das ist vor allem bei der Programmentwicklung nützlich.

```
10 PRINT "Jetzt wird der Drucker eingeschaltet";
20 PRINT CHR$(4); "PR#1"
```

Dieser Befehl wird nicht wie erwartet ausgeführt, wenn voher ein PRINT-Statement mit Komma oder Strickpunkt abgeschlossen wurde. (CHR$(4) wird nicht richtig erkannt).

RUN
Jetzt wird der Drucker eingeschaltetPR#1

Das Problem kann durch ein zusätzliches PRINT-Statement (ohne Komma oder Strichpunkt) bzw. mit HTAB 0 gelöst werden:

```
10 PRINT "Jetzt wird der Drucker eingeschaltet";: HTAB 0
20 PRINT CHR$(4); "PR#1"
RUN
```

Jetzt wird der Drucker eingeschaltet

(Mit Control-Reset oder PR#3 bzw. PR#0 wird wieder der Bildschirm aktiviert).

5.2 Die seriellen Ports des Apple //c

Anders als der Apple //e hat der Apple //c keine physischen Slots. Um Apple //e – Programme, die Slots verwenden, auch für den Apple //c verfügbar zu machen, wurde die Apple //c Firmware so ausgelegt, daß diese Slots durch Software simuliert werden. Die Einsprungadressen sind deshalb auch gleich wie die im Apple //e.

Die beiden seriellen Anschlüsse des Apple //c sind ausgelegt für den Anschluß eines Druckers (Port 1, RS-232) und eines Modems oder Akustikkopplers (Port 2, Serial communications port). Diese Standardeinstellungen beider Ports können durch Software geändert und dem jeweiligen Ausgabegerät angepaßt werden.

Die Standardwerte sind folgende:

PORT 1	PORT 2
9600 Baud	300 Baud
8 Databits	8 Databits
keine Parity	keine Parity
2 Stopbits	1 Stopbit
80 Zeichen/Zeile	Kein Carriage Return
Kein Echo	Kein Echo
Linefeed nach CR	Kein Linefeed nach CR
Kommandos mit CONTROL-I	Kommandos mit CONTROL-A

Manche Programme wie „APPLEWORKS" oder „ACCESS//" erlauben die Änderung dieser Standardwerte vom Programm aus. Mit den ProDOS System Dienstprogrammen (System Utilities) sind Änderungen an den seriellen Ports ebenfalls möglich.

Aus einem Basic- oder Assemblerprogramm oder als Einzelanweisung erfolgen Änderungen dieser Standardwerte mit CONTROL-I (bei Port 1) bzw. CONTROL-A (bei Port 2) gefolgt vom jeweiligen Befehl.
Beispiel:

```
10 PRINT CHR$(9);"8B"
```

schaltet Port 1 (seriellen Ausgang 1) auf 1200 Baud (CHR$(9) = CONTROL-I; 8B ist der Code für 1200 Baud).

5.2.1 Serielle Parts konfigurieren

5.2.1.1 Baudrate

Die Baudrate gibt die Geschwindigkeit der Datenübertragung an. Ein Baud ist 1 Bit pro Sekunde. Bei Computern werden Baudraten zwischen 50 und 19200 verwendet. Am gebräuchlichsten sind 1200 oder 9600 Baud (für Drucker), bzw. 300 Baud (Modem).

Die Baudrate läßt sich beim Apple //c einstellen durch den Befehl:

CONTROL-I nnB bzw.
CONTROL-A nnB (für Port 2)

wobei nn eine Zahl zwischen 1 und 15 ist. Im Einzelnen bedeutet nn:

nn	Baudrate	nn	Baudrate
1	50	9	1800
2	75	10	2400
3	110	11	3600
4	135	12	4800
5	150	13	7200
6	300	14	9600
7	600	15	19200
8	1200		

5.2.1.2 Datenformat

Das Datenformat spezifiziert die Anordnung eines Bytes innerhalb einer Übertragungseinheit.
Eine Übertragungseinheit besteht aus den Bits eines zu übertragenden Bytes und mehreren Kontrollbits. Im Allgemeinen besteht sie aus:

Startbit + Datenbits + Paritybit + Stopbit

Jedes Bit bedeutet ein Signal am entprechenden Pin des seriellen Anschlußes am Computer in Form einer Spannung (entweder ca. 3 Volt oder 0 Volt) während einer gewissen Zeitdauer (z.B. 0,05 Sekunden bei 1200 Baud).
Das Startbit ist bei allen Datenformaten vorhanden.
Die Zahl der Datenbits beträgt je nach Format 7 oder 8 (6 Bits wird sehr selten verwendet); beginnend mit dem niedersten Bit (0).
Das Paritybit ist ein reines Kontrollbit, mit dem der Datenstrom vom Empfänger auf Fehlerfreiheit überprüft werden kann. Das Paritybit existiert in 4 Versionen:

Odd Parity — die Zahl der Einsen unter den Datenbits wird gezählt; ist die Zahl gerade, so wird das Paritybit eins, sonst null.

Even Parity	genau umgekehrt wie Odd Parity; das Bit wird immer so gesetzt, daß die Zahl der Einsen von Datenbits und Paritybit immer gerade ist.
Mark Parity	Paritybit ist immer 1.
Space Parity	Paritybit ist immer 0.

Manchmal wird das Paritybit auch weggelassen (No Parity).

Das Stopbit (manchmal werden auch zwei verwendet) zeigt dem Empfänger das Ende der Übertragungseinheit an.
Das Datenformat und das Paritybit werden beim Apple//c wie folgt eingestellt:

Datenformat:	CONTROL-I nD bzw. CONTROL-A nD bei Port 2

wobei n eine Zahl zwischen 0 und 7 ist. Die Zahl n bedeutet:

n	Datenbits	Stopbits
0	8	1
1	7	1
2	6	1
3	5	1
4	8	2
5	7	2
6	6	2
7	5	2

Paritybit:	CONTROL-I nP CONTROL-A nP	bzw. bei Port 2

n ist eine Zahl zwischen 0 und 7 und bedeutet:

n	Paritybit
0	kein
1	Odd (ungerade)
2	kein

3	Even (gerade)
4	kein
5	Mark (immer 1)
6	kein
7	Space (immer 0)

5.2.1.3 Zeilenbreite

Die Zeilenbreite wird (soweit vorhanden) eingestellt durch:

CONTROL-I nnnN	bzw.
CONTROL-A nnnN	bei Port 2

nnn ist eine Zahl zwischen 1 und 255. Nach nnn Zeichen wird automatisch ein Carriage Return (CR) oder Wagenrücklauf eingefügt. (Um bei Datenübertragungen CR zu unterdrücken, wird Speicherzelle $579 (dez 1401) für Port 1 bzw. $57A (dez 1402) für Port 2 auf 0 gesetzt oder Zap benutzt (siehe Z)).
Das N am Ende des Befehls kann durch ein Carriage Return (CHR$(13)) ersetzt werden.

5.2.1.4 Echo

Als Echo wird die gleichzeitige Ausgabe der über die seriellen Ports (z.B. an einen Drucker) ausgegebenen Daten auf den Monitor bezeichnet. Das Echo ist normalerweise nicht eingeschaltet. Es wird aktiviert durch:

CONTROL-I I	bzw.
CONTROL-A I	für Port 2

Die Verwendung von Echo ist bei der Ausgabe an Drucker praktisch; bei Datenübertragung zu anderen Computern kommt es nach dem ersten Tastendruck zu einer endlosen Schleife, wenn beide Computer Echo verwenden.

5.2.1.5 Zeilenvorschub nach Carriage Return (CR)

Manche Drucker führen automatisch einen Zeilenvorschub aus, wenn sie am Ende einer Zeile ein Carriage Return (Wagenrücklauf) erhalten, andere nicht.

Je nach Drucker muß deshalb ein Zeilenvorschub (Linefeed) (CHR$(10)) nach Carriage Return eingefügt werden, oder nicht.

Zeilenvorschub nach CR wird eingeschaltet durch:

CONTROL-I L bzw.
CONTROL-A L bei Port 2

Zeilenvorschub nach CR wird ausgeschaltet durch:

CONTROL-I K bzw.
CONTROL-A K bei Port 2

5.2.1.6 Standardwerte setzen

Mit dem Befehl R (Reset) werden die oben dargestellten Standardwerte der seriellen Ports 1 und 2 eingestellt:

CONTROL-I R bzw.
CONTROL-A R bei Port 2

5.2.1.7 BREAK

Manche Drucker verwenden ein BREAK Signal zur Synchronisation. Mit dem Befehl

CONTROL-I S bzw.
CONTROL-A S bei Port 2

wird ein BREAK Signal (0,223 sek) ausgegeben.

5.2.1.8 ZAP

Der ZAP Befehl verhindert die Ausführung aller nachfolgender Kontrollbefehle durch die Firmware:

CONTROL-I Z bzw.
CONTROL-A Z bei Port 2

Danach können beliebige Steuerzeichen über den Port ausgegeben werden, ohne daß die Apple Firmware davon Notiz nimmt.

Der ZAP Befehl kann nur mit CONTROL-RESET oder PR#1 (bzw. PR#2 bei Port 2) neutralisiert werden.

5.2.1.9 Andere Kontrollzeichen

Um eventuelle Schwierigkeiten mit den angeschlossenen Geräten durch die Kontrollzeichen CONTROL-I oder CONTROL-A zu vermeiden, kann das Kontrollzeichen geändert werden:

CONTROL-I CONTROL-V

aktiviert CONTROL-V als Kontrollzeichen für Port 1; CONTROL-I hat dann keine Bedeutung mehr. Mit

CONTROL-V CONTROL-I

wird wieder CONTROL-I als Kontrollzeichen aktiviert. Das gleiche gilt mit CONTROL-A für Port 2.
Als Ersatzkontrollzeichen sind CONTROL-V und CONTROL-W vorgesehen. Andere Kontrollzeichen können einen unkontrollierten Systemzustand hervorrufen.

5.2.1.10 Spezifische Port 2 Befehle

Port 2 ist als Kommunikationsport ausgelegt. Dafür sind einige spezielle Befehle vorhanden:

Terminal Modus

CONTROL-A T schaltet den Terminal Modus ein.

CONTROL-A Q schaltet den Terminal Modus aus.

Der Terminal Modus kann auch von externen Geräten ausgelöst werden (vorausgesetzt, der Port 2 ist bereits mit IN#2 aktiviert):

CONTROL-T schaltet von außen Port 2 auf Terminal Modus

CONTROL-R schaltet von außen Terminal Modus ab.

Die aktuellen Einstellungen der Ports bleiben so, bis der Computer ausgeschalten wird. Beim Wiedereinschalten stellen sich wieder die Standardwerte ein. OPEN-APPLE-CONTROL-RESET ändert die Einstellungen nicht.

Die Änderungen können automatisch ausgeführt werden, um z.B. einen Drukker in Port 2 anzusprechen. Dazu wird ein STARTUP Programm unter ProDOS (bzw. ein HELLO Programm unter DOS 3.3) verwendet.

```
10 REM Hello user, my name is STARTUP
20 REM Konfiguriert PORT 2 als Druckerport
30 REM   1200 Baud
40 REM   8 Databits
50 REM   no Parity
60 REM   1 Stopbit
70 REM   160 Zeichen/Zeile
80 REM   Linefeed nach CR
90 REM
100 PRINT CHR$(4) ;"PR#2"
110 PRINT CHR$(1) ;"8B" : REM - 1200 BAUD
120 PRINT CHR$(1) ;"0D" : REM - 8 DATABITS, 1 STOPBIT
130 PRINT CHR$(1) ;"0P" : REM - NO PARITY
140 PRINT CHR$(1) ;"160N";: REM - 160 CPL
150 PRINT CHR$(1) ;"L";: REM - ADD LINEFEED AFTER CR
160 PRINT CHR$(4) ;"PR#3"
170 PRINT "PORT 2 ist Druckerausgang"
180 NEW
```

Die Hardware Page Locations der ACIA sind im Anhang aufgeführt.
ACIA = Asynchronus Communication Interface Adapter. Die Bezeichnung ACIA wird für die seriellen I/O Chips (6551) des Apple //c verwendet.

Kapitel 6

Weiterführende Informationen

Dem Benutzer stehen als Informationsquellen in der Regel nur Bücher und Zeitschriften zur Verfügung. Die Computerzeitschriften, welche sich in nennenswertem Umfang und Niveau mit Computern der Apple // Familie beschäftigen, sind unten dargestellt.

Die Firma Apple bietet eine Reihe von Manuals für den Apple //c, Betriebssysteme und Sprachen an. Bedauerlicherweise ist die Information darin so gegliedert, daß sehr starke Überschneidungen auftreten und teilweise nutzlose oder unvollständige Information geboten wird; gleichzeitig sind die Preise für diese Manuals sehr hoch. Ein Teil des Materials ist nur in englischer Sprache erhältlich.
Weiter wird eine Vielzahl von Büchern unterschiedlicher Aktualität und Thematik angeboten. Eine Reihe dieser Bücher sind unten dargestellt. Englischsprachige Bücher wurden nicht in die Liste aufgenommen.

Oft sind Detailinformationen, vor allem Neuigkeiten und Tricks in Computerclubs zu erfahren, die auch zum Softwaretausch sehr beliebt sind. Adressen sind in einschlägigen Zeitschriften oder beim Händler erhältlich.

6.1 Bücherverzeichnis

Titel — Autor
Verlag — Ort — Jahr — ISBN-Nr.
Kommentar

Basic und Betriebssysteme:

Apple // Anwenderhandbuch — L. Pool
te-wi — München — 1985 — 3-921803-52-7
Darstellung der Apple // Computer

Apple //c Reference Manual
Apple — 1984
Hardware + ROM Beschreibung (engl.)

Programme für meinen Apple — S.R.Trost
SYBEX — Düsseldorf — 1983 — 3-88745-029-9
Mathem./ wirtschaftl. Applesoft Basic Programme

Applesoft BASIC Tips und Tricks — Frank Bühler
Hüthig — Heidelberg — 1985 — 3-7785-1094-0
Applesoft Basic mit vielen Beispielen

APPLESOFT-BASIC — W. Dederichs
BI — Mannheim — 1982 — 3-411-00603-X
Einführung in Applesoft Basic

BASIC Übungen für den Apple — J.P. Lamoitier
SYBEX — Düsseldorf — 1983 — 3-88745-016-7
Viele praktische und sinnvolle Übungen

Diverse Manuals
Apple
Basic-, Pascal-, DOS Manuals

Apple DOS 3.3 — Ulrich Stiehl
Hüthig — Heidelberg — 1984 — 3-7785-1049-5
Einführung in das Betriebsystem DOS 3.3

Titel Autor
Verlag Ort Jahr ISBN-Nr.
Kommentar

ProDOS für Aufsteiger II Ulrich Stiehl
Hüthig Heidelberg 1985 3-7785-1036-3
Weiterführendes Buch über ProDOS, Hilfsprogramme

ProDOS für Aufsteiger I Ulrich Stiehl
Hüthig Heidelberg 1984 3-7785-1027-4
Beschreibung des Betriebssystems ProDOS

Die ProDOS Analyse Arne Schäpers
Hüthig Heidelberg 1985 3-7785-1134-3
Analyse des ProDOS für fortgeschr. Assemblerprogrammierer

Grafik:

Apple // Raster Grafik J. Stanton
Pandabooks Berlin 1984 3-89058-006-8
Grafikprogramme in Basic und Assembler

Bewegte Apple Grafik Arne Schäpers
Hüthig Heidelberg 1985 3-7785-1150-5
Schnelle Spielegrafik mit HRCG und Assembler

Apple Pascal Grafik Tom Swan
Pandabooks Berlin 1984 3-89058-009-2
Grafikprogramme in Pascal

Mikrocomputer Grafik Roy. E. Myers
Pandabooks Berlin 1983 3-89058-000-9
Gute Darstellung 2D/3D Grafik für Apple //
Vorwiegend in Applesoft Basic

Titel Autor
Verlag Ort Jahr ISBN-Nr.
Kommentar

Assembler, Maschinensprache:

6502 Programmiern in Assembler L.A.Leventhal
te-wi München 1981 3-921803-10-1
Umfassende Darstellung des 6502 Assemblers

6502/65C02 Maschinensprache C. Persson
Heise 1983 3-992705-20-0
Darstellung der 6502 und 65C02 Befehle am Beispiel des Elrad Cobold-Computers

Programmierung des 6502 R. Zaks
SYBEX Düsseldorf 1982 3-88745-011-6
Einführung in die 6502 Befehle, Neuauflage auch 65C02

Apple-Assembler lernen Bd.1 Jürgen B. Kehrel
Hüthig Heidelberg 1985 3-7785-1151-3
Einführung für 6502 und 65C02, mit viel Software auf Begleitdiskette (Assembler, Simulator)

Apple // Assemblerprogrammierung Roger Wagner
Pandabooks Berlin 1984 3-89058-003-3
Einführung in Apple // Assembler

Apple Assembler Ulrich Stiehl
Hüthig Heidelberg 1984 3-7785-1047-9
Apple Assembler Routinen und Programme

Apple Maschinensprache Inman/Inman
te-wi München 1984 3-921803-21-7
Einführung in Apple // Assembler von Basic aus

Titel Autor
Verlag Ort Jahr ISBN-Nr.
Kommentar

Andere Sprachen:

PASCAL in 100 Beispielen Ottmann
Teubner Stuttgart 1983 3-519-02515-9
Programmbeispiele in Apple Pascal

Apple // Pascal Luehrmann
te-wi München 1982 3-921803-04-7
Einführung in Apple Pascal

Programmieren in FORTH Leo Brodie
Hanser München 1984 3-446-14070-0
Einführung in die Sprache FORTH

6.2 Zeitschriften

PEEKER, Magazin für Apple Computer
MC, die Mikrocomputer Zeitschrift
c't, Magazin für Computertechnik
A+, Englischsprachig
BYTE, Englischsprachig
NIBBLE, Englischsprachig
ORCHARD, Englischsprachig

Anhang 1

BASIC und Betriebssysteme

Anweisungen und Funktionen

Die Anweisungen sind alphabetisch geordnet. Neben der Bezeichnung sind Beispiele für die Anwendung gegeben und die Anweisung ist kurz erläutert.

ABS(Zahl) z.B. ABS(-44.3)

ermittelt den absoluten Betrag der Zahl.

APPEND Filename z.B. 100 PRINT CHR$(4);"APPEND SEQDAT4"
Öffnet ein sequentielles File auf der Diskette wie OPEN. Die mit WRITE auf das File geschriebene Information wird im Anschluß an die schon vorhandene Information gespeichert. APPEND kann nur innerhalb von Programmen mit einleitendem Control-D verwendet werden.

ASC(Zeichen) z.B. ASC(a)
ermittelt den ASCII-Wert des angegebenen Zeichens (siehe ASCII Tabelle).

ATN(Zahl) z.B. ATN(2)
ermittelt den Arcustangens des in Radian angegebenen Winkels.

BLOAD Filename z.B. BLOAD Xref,A768 BLOAD Datap
BLOAD liest binäre Daten von einem File ein, zum Beispiel ein Maschinenprogramm. Die Daten sind anschließend im RAM an der Stelle vorhanden, von der aus sie in das File geschrieben wurden. Durch den Zusatz ,A(Startadresse) kann die Startadresse der Daten im RAM geändert werden. Die Daten, die vorher im RAM vorhanden waren, gehen verloren.
Die Start- oder Anfangsadresse der mit DOS 3.3 eingelesenen Daten steht in Adresse AA72,AA73 hex, die Länge in Adresse AA60,AA61 hex.

BRUN Filename z.B. BRUN FID BRUN FLIP,A$200
Liest wie BLOAD binäre Daten ins RAM und führt diese als Maschinenprogramm aus. Die Startadresse kann mit dem Zusatz ,A(Startadresse) gewählt werden.

BSAVE Filename,A(Startadresse),L(Länge)
z.B. BSAVE FLIP,A768,L141 BSAVE FILP,A$300,L$8D
Schreibt Binärdaten aus dem RAM von Startadresse bis (Startadresse + Länge) auf ein Binärfile. Startadresse und Länge können dezimal oder hexadezimal (mit $-Zeichen) angegeben werden.

CALL Adresse z.B. CALL -151
Ruft ein Unterprogramm in Maschinensprache an der angegebenen Adresse auf. Siehe Anhang.

CATALOG ,D(Laufwerksnummer) (unter ProDOS auch CAT)
z.B. CATALOG ,D1
Gibt Inhaltverzeichnis des aktuellen Diskettenlaufwerks an. Darstellung ist vom Betriebssystem abhängig.

CHAIN Filename z.B. CHAIN Prog12 (nur unter ProDOS)
liest ein Basic-Programm von der Diskette und startet es, ohne bisher verwendete Programmdaten zu zerstören.

CHR$(Zahl) z.B. CHR$(4)
ermittelt das Zeichen mit dem angegebenen ASCII Code (0 bis 255).

CLEAR z.B. CLEAR 100 PRINT CHR$(4);"CLEAR"
Löscht alle Variablen eines Programmes. Nach dem Aufruf aus einem Programm läuft dieses anschließend weiter.

CLOSE (Filename) z.B. CLOSE FLIP 100 PRINT CHR$(4);"CLOSE"
Schließt File(s) auf der Diskette. Wurden in einem File mit WRITE Information gespeichert, so muß CLOSE verwendet werden, um alle Informationen abzuspeichern. CLOSE ohne Angabe des Filenamens schließt alle Files, außer EXEC Files.

COLOR= z.B. COLOR=12
Bestimmt die Farbe für anschließend gezeichnete Bildpunkte in niedrig auflösender Grafik.

Farbe	Nummer	Farbe	Nummer
schwarz	0	braun	8
magenta	1	orange	9
dunkelblau	2	grau	10
purpur	3	rosa	11
dunkelgrün	4	hellgrün	12

grau	5	gelb	13
blau	6	aquamarin	14
hellblau	7	weiß	15

CONT
Bewirkt die Fortsetzung eines Programmes, welches mit Control-C, STOP oder END abgebrochen wurde. Beim Abbruch während eines INPUT's oder nach Änderungs des Programms erfolgt ein CAN'T CONTINUE ERROR.

COS(Zahl) z.B. COS(2)
ermittelt den Cosinus des in Radian angegebenen Winkels.

CREATE Filename/Directoryname (nur unter ProDOS)
z.B. CREATE /DISK3/SUBRAM/
Erstellt ein File oder Subdirectory mit dem angegebenen Namen auf der Diskette.

DATA z.B. 100 DATA 12,14,93548,4
100 DATA ESSEN,12,"BADEN BADEN",-45
Speichert eine Reihe von Informationen (Zahlen oder Texte), die durch Kommas getrennt sind, ab. Die Informationen können durch wiederholtes READ wieder gelesen werden. Texte die Kommas, Strichpunkte oder Freizeichen enthalten, müssen in Anführungszeichen stehen. DATA ist nur innerhalb von Programmen sinnvoll.

DEF FN Funktionssname z.B. 100 DEF Quadrat(X)= X`
Innerhalb von Programmen wird eine Funktion definiert. DEF muß alleine in einer Programmzeile stehen. Die Definition kann zuvor definierte Funktionen enthalten. Der in der Definition angegebene Variablenname ist nicht bindend.

DEL Programmzeile,Programmzeile
z.B. DEL 200,299 100 DEL 200,299
Löscht bestehende Programmzeilen innerhalb des angegebenen Bereichs (einschließlich), soweit vorhanden. Bei der Ausführung innerhalb eines Programm wird dieses anschließend abgebrochen.

DELETE Filename z.B. DELETE FLIP
100 PRINT CHR$(4);"DELETE FLIP"
Markiert ein File im Inhaltsverzeichnis der Diskette als gelöscht.

DIM

z.B. DIM A$(20) 100 DIM B%(100,10,10,10),C(55),S$(20)

Reserviert Speicherplatz für Felder von Variablen. Mehrfache Dimensionierung der gleichen Variablen ist nicht zulässig. Bei Verwendung nicht dimensionierter Feldvariablen wird automatisch ein Feld der Ausdehnung 0–10 in jeder verwendeten Richtung dimensioniert.

DRAW z.B. DRAW 1 AT 110,33
Zeichnet die zuvor definierte Linienfolge 1 an der Stelle Spaltennr,Zeilennr in hochauflösender Grafik.

END z.B. 100 END
Bewirkt Abbruch des Programms in der betreffenden Zeile. Es ist nicht notwendig, wenn END in der letzte Zeile steht.

EXEC Filename
z.B. EXEC Protext 100 PRINT CHR$(4);"EXEC Protext"
Liest den Inhalt eines Textfiles auf der Diskette ein und interpretiert ihn wie eine Tastatureingabe. Der Inhalt kann Programmzeilen enthalten, die dann gespeichert, aber nicht ausgeführt werden; Anweisungen an das Betriebssystem werden sofort ausgeführt (z.B LOAD, RUN etc.). Das EXEC-File wird automatisch nach dem letzten Befehl oder nach einem im Inhalt enthaltenen weiteren EXEC Befehl geschlossen.

EXP(Zahl) z.B. EXP(10)
errechnet den Wert der Funktion e hoch Zahl (e=2.71828183)

FLASH
Bewirkt die blinkende Darstellung der nachfolgend ausgegeben Zeichen auf dem Bildschirm.

FLUSH Filename z.B. FLUSH DAT3 (nur unter ProDOS)
leert den I/O Puffer des angegebenen Files.

FN Name(Zahl) z.B. FN Wurzel(4)
errechnet den Wert der mit DEF FN zuvor definierten Funktion an der angegebenen Stelle.

FOR ... TO ...
z.B. FOR I=0 TO 14 100 FOR B%=100 TO 10 STEP -3
beginnt eine Schleife zwischen dem FOR ... TO Befehl und dem dazugehörigen NEXT. Die Zählvariable wird nach jedem Durchgang um eins erhöht oder um den nach dem STEP angegebenen Betrag geändert. FOR NEXT Schleifen können maximal in 10 Ebenen geschachtelt werden.

FP (DOS3.3)
schaltet (von Integer Basic) auf Applesoft Basic, zerstört dabei Programm und Daten.

FRE(0)

ermittelt die Zahl der noch (für Basic) freien Speicherplätze. FRE(0) berücksichtigt nur die aktuelle 64 kB Bank. FRE(0) löst die „Garbage Collection" aus, mit der unbenötigte Strings entfernt werden.Die „Garbage Collection" kann auch mit dem wesentlich schnelleren ProDOS Befehl FRE aufgerufen werden.

GET z.B. GET X$
liest ein Zeichen vom aktellen Eingabegerät, normalerweise der Tastatur, ohne Wiedergabe des Zeichens auf dem Bildschirm. Die Eingabe eines unzulässigen Zeichens, z.B. eines Buchstabens statt einer Zahl bei GET A bewirkt einen SYNTAX ERROR.

GOSUB z.B. GOSUB 100
bewirkt den Sprung in eine Subroutine. Der Rücksprung erfolgt nach dem Befehl RETURN. GOSUB Befehle können in maximal 24 Ebenen geschachtelt werden.

GOTO z.B. GOTO 100
bewirkt den Sprung in die angegebene Zeile. Ist die Zeile nicht vorhanden, so erscheint die Fehlermeldung „UNDEFINED STATEMENT ERROR".

GR
schaltet auf niedrigauflösende Grafik (40 x 40 Zeichen); die untersten 4 Zeilen des Bildschirms bleiben für die Textausgabe reserviert. Mit POKE-16302,0 wird der ganze Bildschirm für Grafik (40 x48) verwendet, mit POKE-16301,0 werden wieder die 4 untersten Zeilen auf Textausgabe geschaltet.

HCOLOR= z.B. HCOLOR=7
bestimmt die Farbe, in der die nachfolgenden Darstellungen in hochauflösender Grafik ausgeführt werden.

Farbe	Nummer	Farbe	Nummer
schwarz	0	schwarz	4
grün	1	orange	5
violett	2	blau	6
weiß	3	weiß	7

HGR
schaltet auf hochauflösende Grafik (280 x 160), die 4 untersten Zeilen bleiben für die Textausgabe reserviert. Der Inhalt des Bildschirms bzw. des dazugehörigen Speicherbereichs (8192 bis 16383) wird dabei gelöscht. (siehe auch PEEK und POKE Adressen im Anhang).

HGR2
schaltet auf hochauflösende Grafik von Speicherbereich 2 (280 x 192). Der Inhalt des Bildschirms bzw. des dazugehörigen Speicherbereichs (163844 bis 24575) wird dabei gelöscht. (siehe auch PEEK und POKE Adressen im Anhang).

HIMEM: z.B. HIMEM:38000
setzt den oberen Grenzwert des für Basic-Programme und deren Daten zugänglichen Speicherberiechs auf den angegeben Wert. Normalerweise steht der HIMEM: Zeiger selbsttätig auf 38400.

HLIN Spaltenr,Spaltenr AT Zeilenr
z.B. HLIN 12,34 AT 18
zeichnet in niedrigauflösender Grafik eine horizontale Linie in der zuvor definierten Farbe in Zeilenr von Spaltenr bis Spaltenr.

HOME
löscht den Textbildschirm und setzt den Cursor links in die erste Zeile.

HPLOT
z.B. HPLOT 14,110 HPLOT TO 200,44 HPLOT 100,100 TO 110,120
HPLOT 1,1 TO 1,10 TO 10,10 TO 10,1 TO 1,1
zeichnet eine Linie in hochauflösender Grafik in der zuvor definierten Farbe. Wird kein TO Kommando verwendet, so wird nur ein Punkt gezeichnet. Werden vor dem TO Kommando kein Anfangswerte angegeben, so wird die Linie von der zuletzt gezeichneten Stelle aus gezogen. In einer HPLOT Anweisung können mehrere TO Kommandos nacheinander verwendet werden.

HTAB z.B. HTAB 12
setzt den Cursor bei Textausgabe in die angegebene Spalte in der aktuellen Zeile.

IF ... THEN
z.B. IF A=3 THEN 200 IF Z$="JA" THEN PRINT ...
IF X=0 GOTO 100

führt die in der gleichen Zeile stehenden Anweisungen aus, wenn der Vergleich zwischen IF und THEN wahr ist. IF ... kann in den Formen IF ... THEN, IF ... GOTO und IF ... THEN GOTO verwendet werden. Zwischen IF und THEN können mehrere Vergleiche durchgeführt werden, die durch AND oder OR verknüpft sind.

IN# z.B. IN#2 100 PRINT CHR$(4);"PR#2"
schaltet die Eingabe auf den angegebenen Slot. Normalerweise ist die Tastatur aktiviert (IN#0).

INIT (nur DOS 3.3) z.B. INIT Hello
initialisiert (formatiert) eine Diskette im DOS 3.3 Format. Ein im RAM-Speicher vorhandenes Programm wird unter dem angegebenen Namen auf der Diskette gespeichert.

INPUT
z.B. INPUT X$ INPUT X$,A INPUT "Bitte RETURN drücken"; X$
erwartet eine oder mehrere Eingaben, die durch RETURN oder Kommas getrennt sind, und ordnet diese den Eingabevariablen zu. Werden statt einer erwarteten numerischen Eingabe unzulässige Zeichen eingegeben, so erfolgt die Fehlermeldung „REENTER" und der Rechner erwartet erneut die verlangte Eingabe. Wird kein in Anführungszeichen gestellter Hinweis dargestellt, so wird von INPUT ein Fragezeichen ausgegeben. Ein Hinweis wird von der Eingabevariablen durch einen Strichpunkt getrennt

INT(Zahl) z.B. INT(13.44) (Applesoft Anweisung)
rundet ab, ermittelt die nächstkleinere (oder gleiche) ganze Zahl.

INT (DOS 3.3 Anweisung)
schaltet (von Applesoft Basic) auf Integer Basic.

INVERSE
bewirkt die invertierte Darstellung dunkle Schrift auf hellem Hintergrund von Zeichen bei der Ausgabe von Text.

**LEFT$**(String,L) z.B. LEFT$(A$,3)
ermittelt den linken Teil von String mit Länge L.

LEN(String) z.B. LEN(X$)
ermittelt die Länge des angegebenen Strings.

LET z.B. LET A=3
Der Zahlenwert wird der angegebenen Variablen zugewiesen. Die Zuweisung ist auch ohne LET möglich.

LIST
z.B. LIST LIST ,100 LIST 100, LIST 100,200
listet die im RAM-Speicher vorhandenen Programmzeilen innerhalb der optional angegebenen Grenzen. Werden eine oder beide Grenzen nicht angegeben, so wird die erste bzw. letzte Zeile bzw. beide verwendet.

LOAD Filename
z.B. LOAD Dezhex 100 PRINT CHR$(4);"LOAD DEZHEX"
liest ein BASIC Programm von der Diskette in den Speicher. Bei der Verwendung innerhalb von Programmen muß LOAD als PRINT Anweisung mit Ctrl-D verwendet werden. Bei der Ausführung von LOAD wird ein vorher im Speicher vorhandenes Programm inklusive Daten gelöscht.

LOCK z.B. LOCK DEZHEX
markiert ein File auf der Diskette als geschützt. Ein solches File kann weder überschrieben noch gelöscht werden. Der Schreibschutz wird mit UNLOCK wieder aufgehoben.

LOG(Zahl) z.B:. LOG(10)
ermittelt den natürlichen Logarithmus (zur Basis e) der angegebenen Zahl.

LOMEM: z.B. LOMEM: 16384
grenzt den für BASIC Programme und deren Daten verfügbaren Speicherbereich nach unten ab. Normalerweise ist LOMEM: (ohne Programm) gleich 2048. LOMEM: wird durch Abänderung von Basicprogrammen im Speicher wieder auf den niedrigst möglichen Wert gesetzt. Der Wert von LOMEM: kann durch
PRINT PEEK(105)*256+PEEK(106) ermittelt werden.

MAXFILES (nur unter DOS 3.3) z.B. MAXFILES 6
Dient zur Reservierung von je 595 Byte Puffer für offene Files. MAXFILES wird von DOS 3.3 auf 3 gesetzt und kann vom Benutzer zwischen 0 und 16 variiert werden.

MON C,I,O (nur unter DOS 3.3)
dient zur Darstellung der Daten beim Diskettenzugriff, C (Commands),I (Inputdaten) und O (Outputdaten) können wahlweise verwendet werden.

MID$(String, N, L) z.B. MID$(A$,5,8)
liefert einen Teil vom Text-String ab Buchstabe N mit Länge L. Die Länge L kann auch weggelassen werden.

NEW
löscht ein vorhandenes BASIC-Programm und Daten aus dem Speicher; verändert LOMEM:.

NEXT Variablenname z.B. NEXT I% 100 NEXT A,B,C%
markiert das Ende einer FOR...NEXT Schleife. Der Variablenname kann weggelassen werden. Mehrere Schleifen können durch Angabe der Variablennamen getrennt durch Kommas direkt nacheinander beendet werden.

NOMON C,I,O
schaltet die mit MON C,I,O aktivierte Ausgabe von Diskettenzugriffsdaten aus. C,I,O sind wahlweise verwendbar.

NORMAL
schaltet INVERSE oder FLASH Textdarstellung aus.

NOTRACE
schaltet die mit TRACE aktivierte Ausgabe von durchlaufenen Zeilennummern aus.

ONERR GOTO Zeilennummer z.B. ONERR GOTO 100
dient zur Fehlerbehandlung, bewirkt beim Auftreten eines Fehlers im Programm einen Sprung zu Zeilennummer. Mit PEEK(222) kann die Fehlernummer ermittelt werden (siehe Anhang). Die interne Fehlerroutine des Apple wird nicht immer korrekt ausgeführt. Die Routine arbeitet einwandfrei, wenn bei jeder Fehlerbehandlung das folgende Maschinenprogramm ausgeführt wird:
hex:68 A8 68 A6 DF 9A 48 98 48 60
dez:104 168 104 166 223 154 72 152 72 96
Dieses Programm kann (z.B. mit POKEs) gespeichert werden (z.B. ab dez 768 od. hex 300) und bei jeder Fehlerbehandlung mit CALL (z.B. 768) ausgeführt werden.

ON ... GOSUB z.B. 400 ON A GOSUB 100,200,300
verzweigt nach einer Zeilennummer, je nach Wert der Variablen A (0 < = A < = 255). Bei A gleich 0 oder einem Wert, dem keine Zeilennummer zugeordnet ist, wird der Sprungbefehl nicht ausgeführt.

ON ... GOTO z.B. 200 ON X% GOTO 1000,300,20
verzweigt nach einer Zeilennummer, je nach Wert der Variablen X% (0 < = X% < = 255). Bei X% gleich 0 oder einem Wert, dem keine Zeilennummer zugeordnet ist, wird der Sprungbefehl nicht ausgeführt.

OPEN Filename
z.B. 100 PRINT CHR$(4); "OPEN TEXTFILE"
110 PRINT CHR$(4); "OPEN DAFIL,L100"
öffnet das File „Filename" auf der Diskette. Vor dem Datenzugriff muß das File mit READ oder WRITE als Input oder Outputfile markiert werden. Ist das File bei der Ausführung von OPEN nicht vorhanden, so wird ein solches (leeres) File erzeugt.
Bei der Angabe der Recordlänge (z.B. ,L130) nach dem Filenamen wird ein File für Direktzugriff eröffnet, ohne Recordlänge wird ein sequentielles File eröffnet. Sequentielle Files können auch ohne den OPEN Befehl verwendet werden.

PDL z.B. 100 PRINT PDL(1)
ermittelt den Wert eines Analogsignals (0 bis 3) am Mauseingang.

PEEK z.B. X= PEEK (222)
ermittelt den Wert in der dezimal angegebenen Speicherzelle.

PLOT Spaltennr, Zeilennr z.B. PLOT 15,14
zeichnet einen Punkt in niedrigauflösender Grafik in Spalte Spaltennr. und Zeile Zeilennr.

POKE Adresse, Wert z.B. POKE 50,63
speichert in der dezimal angegebenen Adresse den dezimalen Wert (0 bis 255).

POP
löscht die letzte durch ein GOSUB gespeicherte Rücksprungadresse aus dem RETURN-Stack. POP muß immer dann ausgeführt werden, wenn aus einem GOSUB Unterprogramm z.B: mit GOTO .. hinausgesprungen wird, sodaß das zu diesem GOSUB gehörige RETURN nicht mehr ausgeführt wird.

POS z.B. POS(0)
ermittelt die momentane Spaltenposition des Cursors.

POSITION Filename, Rnummer
z.B. 100 PRINT CHR$(4); "POSITION DATFILE,R177"
setzt den Zeiger eines sequentiellen Files auf die hinter R angegebene Recordnummer.

PR# z.B. PR#1
setzt die Ausgabe auf die angebene Nummer (0 bis 7). Beim Apple //c sind folgende Ausgabegeräte unter diesen Nummern erreichbar.

Nummer	Gerät
0	40 Zeichen Darstellung Bildschirm
1	serieller Ausgang 1, Drucker
2	serieller Ausgang 2
3	80 Zeichen Darstellung Bildschirm
4	Maus
5	nicht belegt
6	eingebautes Diskettenlaufwerk
7	externes Diskettenlaufwerk

Bei PR#6 (bzw. PR#7) wird vom angegebenen Diskettenlaufwerk gebootet.

PREFIX z.B. PREFIX/RAM/ (nur unter ProDOS)
setzt den Prefix auf die angegebene Disk (optional Subdirectory).

PRINT
z.B. PRINT A$,B$,C PRINT X;" DM", PRINT "Hallo"
bewirkt die Ausgabe der angegebenen Variablen und/oder von Text. Die Variablen/Texte können durch Kommas oder Strichpunkt getrennt werden. Steht nach der Variablen/Text ein Strichpunkt, so erfolgt die Ausgabe eines Zeichens direkt anschließend, bei einem Komma in einer festgelegten Spalte, sonst in der nächsten Zeile. Das Wort PRINT kann durch das Fragezeichen abgekürzt werden.

READ Filename (DOS 3.3 / ProDOS Anweisung)

```
z.B.    100 PRINT    CHR$(4);"READ F1$"
        100 PRINT    CHR$(4);"READ F1$,R14,B3"
```

Markiert ein File als Eingabefile, von dem mit INPUT oder GET Daten gelesen werden können. Ist keine Recorndnummer (z.B. ,R14) nach dem Filenamen angegeben so wird ein sequentielles File eröffnet, ansonsten erfolgt der Lesevorgang bei der angegebenen Recordposition. Zusätzlich kann die Bytenummer (z.B. ,B3) angegeben werden von der aus gelesen werden soll. Wird durch nachfolgendes INPUT oder GET über das Fileende hinaus eingelesen, so erfolgt die Meldung „END OF DATA".

READ Variable (Applesoft Anweisung) z.B. 100 READ A$
ordnet eine vorher mit DATA gespeicherte Information der angegebenen Variablen zu. Durch wiederholtes READ werden mehrere gespeicherte Informationen zugeordnet. Ist keine Information vorhanden, so erfolgt eine „OUT OF DATA" Fehlermeldung. Entspricht die Information nicht dem angegebenen Variablentyp (z.B. Text bei READ A) so erfolgt ein „SYNTAX ERROR".

REM z.B. REM – Unterprogramm Wurzel
erlaubt die Eingabe von Kommentaren innerhalb von Programmen (vom REM Statement bis zum Zeilenende).

RENAME Filename1, Filename2
z.B. RENAME HALLO,HELLO 100 PRINT CHR$(4);"RENAME F122,F113"
Das File Filename1 wird in Filename2 umbenannt.

RESTORE (Applesoft Anweisung)
setzt den Zeiger für die nächste READ-Anweisung auf die erste mit DATA gespeicherte Information.

RESTORE Filename (nur unter ProDOS)
z.B. RESTORE Progdat
liest sinngemäß die mit STORE Filename unter ProDOS gespeicherten Programmdaten wieder ein.

RESUME
bewirkt nach einer Fehlerbehandlung mit ONERR GOTO die Fortführung des Programmes an der Stelle, an der der Fehler auftrat.

RETURN
bewirkt den Rücksprung zu dem zuletzt ausgeführten GOSUB Befehl, falls die Rücksprungadresse nicht mit POP gelöscht wurde.

RIGHT$(String,L) z.B. RIGHT$(A$,4)
liefert den rechten Teil mit L Zeichen des angebenen Strings.

RND(Zahl) z.B. RND(1)
liefert eine Zufallszahl im Bereich von 0 bis zur angegebenen Zahl.

ROT= z.B. ROT=16
bewirkt die Drehung der mit DRAW oder XDRAW gezeichneten Elemente um den angebenenen Betrag (0 bis 255) mal 5,625 Grad (90 Grad/16) im Uhrzeigersinn.

RUN Filename z.B. RUN HELLO
liest ein Basic-Programm mit dem angebenen Filenamen von der Diskette und führt es aus. Ein vorher im Speicher vorhandenes Programm wird mit den Daten gelöscht.

RUN Zeilennr z.B. RUN 100 RUN
führt ein im Speicher vorhandenes Programm ab Zeilennr aus.

SAVE Filename z.B. SAVE HELLO
speichert ein Basic-Programm unter Filename auf Diskette ab. Ein schon vorhandenes Basic-File wird überschrieben, wenn es nicht schreibgeschützt ist. Existiert ein File anderen Typs mit gleichem Namen, so erfolgt die Fehlermeldung „FILE TYPE MISMATCH".

SCALE= z. B. SCALE=2
definiert den Vergrößerungsfaktor (0 bis 255) für Darstellungen in hochauflösender Grafik, die mit DRAW oder XDRAW erzeugt werden.

SCRN(Spalte,Zeile) z.B. SCRN(22,22)
ermittelt die Farbe des mit Spalte,Zeile angegebenen Punktes in niedrigauflösender Grafik.

SIGN(Zahl) z.B. SIGN(-34)
ermittelt das Vorzeichen der angegebenen Zahl.

SIN(Zahl) z.B. SIN(2)
ermittelt den Sinus des in Radian angegebenen Winkels.

SPC(Zahl) z.B.SPC(4)
erzeugt in PRINT-Anweisungen die angegebene Zahl Leerzeichen.

SPEED= z.B. SPEED=200
bestimmt Anzahl der Durchläufe durch Verzögerungsschleife (255 – angegebenen Wert). Der Wert muß zwischen 0 und 255 liegen.

SQR(Zahl) z.B. SQR(4)
ermittelt die Quadratwurzel der angegebenen Zahl.

STOP
hält ein Programm an und gibt die Meldung BREAK IN Zeilennummer aus.

STORE Filename z.B. STORE Progdat (nur unter ProDOS)
speichert die aktuellen Programmdaten unter Filename.

**STR$**(Zahl) z.B. STR$(-1.456E-5)
wandelt die angegebene Zahl in einen String um.

TAB Spaltennr z.B. TAB 20
Die Funktion rückt den Cursor in die angegebene Spalte.

TAN(Zahl) z.B. TAN(2)
ermittelt den Tangens des in Radian angegebenen Winkels.

TEXT
schaltet von Grafikmodus auf Textausgabe um. Öffnet das Textfenster.

TRACE
gibt während der Ausführung eines Basic-Programms pro Befehl die aktuelle Zeilennummer an. TRACE wird mit NOTRACE ausgeschaltet.

UNLOCK Filename z.B. UNLOCK HELLO
hebt den Schreibschutz eines Files auf der Diskette auf.

USR(Zahl)
ruft ein Unterprogramm in Maschinensprache auf, zu dem ein Sprungbefehl in den Speicherzellen 10,11,12 steht.

VAL(String) z.B. VAL(A$)
ermittelt den numerischen Wert des angegebenen Strings.

VERIFY Filename z.B. VERIFY HELLO
überprüft die vorher durch SAVE, BSAVE oder PRINT auf Diskette geschriebenen Daten (Prüfsummenkontrolle). Liegt ein Fehler vor, so erscheint ein I/O ERROR, andernfalls erscheint keine Meldung (nur DOS 3.3).

VLIN Spalte1,Spalte2 AT Zeile z.B. VLIN 2,14 AT 20
erzeugt in niedrigauflösender Grafik eine senkrechte Linie von Spalte1 bis Spalte2 in der angegebenen Zeile.

VTAB z.B. VTAB 20
setzt den Cursor in die angegebene Zeile (1 bis 24) in der bisherigen Spalte.

WAIT Speicherplatz, Maske, Wert z.B. WAIT 3000,130,200
hält ein Programm an und vergleicht bitweise Speicherplatz mit Wert; dabei werden nur die Bits berücksichtigt, die in Maske gleich 1 sind. Trifft der Vergleich zu, so wird das Programm fortgesetzt, andernfalls kann die Schleife ohne Ausgang mit Ctrl-RESET verlassen werden.

WRITE Filename,Rnummer,Bbyte
z.B. 100 PRINT CHR$(4);"WRITE Datfil,R120,B6"
öffnet das File Filename zum Schreiben. Anschließend werden alle Datenausgaben (z.B. PRINT) auf das File geleitet. Wird die Recordlänge (nach ,R) nicht angegeben, so wird das File als sequentiell behandelt. Andernfalls wird ab Record nummer und Byte byte auf das File geschrieben.

XDRAW shape AT Spalte,Zeile XDRAW 2 AT 19,39
zeichnet die vorher definierte Linienfolge shape in hochauflösender Grafik beginnend an Spalte,Zeile. Drehung, Vergrößerung und Farbe müssen mit ROT=, SCALE= und HCOLOR= vorher definiert werden.

Anhang 2

Kurzzusammenfassung der Betriebssystemanweisungen für DOS3.3 und PRODOS

Verwendete Abkürzungen:

f = Filename (DOS oder PRODOS)
p = Filename (PRODOS)
,S Slotnummer
,D Laufwerknummer
,V Diskettennummer
,R Recordnummer
,A Anfangsadresse
,L Länge

Bezeichnung	Beispiel	Betriebsystem
APPEND f,S,D,V	APPEND DATFIL,D1 Öffnet File, setzt Zeiger auf Fileende.	DOS PRODOS
BLOAD f,A,L,S,D,V	BLOAD PROG.BIN,D1 Liest Binärfile ein.	DOS PRODOS
BRUN f,A,L,S,D,V	BRUN PROG.BIN liest und startet Binärfile.	DOS PRODOS

Bezeichnung	Beispiel	Betriebsystem
BSAVE f,A,L,S,D,V	BSAVE PROG.BIN,A$300,L$74 speichert Binärfile auf Disk.	DOS PRODOS
CATALOG	CATALOG Inhaltsverzeichnis der Disk.	DOS PRODOS
CAT	CAT Sonderform von CATALOG für 40Z/Z.	PRODOS
CHAIN f,S,D,V	CHAIN PROGDAT.2,D1 Liest und startet neues Programm von Disk. Programmdaten werden nicht gelöscht. Unter DOS nur mit INT-Basic.	DOS PRODOS
CLOSE F,S,D,V	CLOSE TEXTDAT,D2 Schließt File(s).	DOS PRODOS
CREATE p,S,D	CREATE V4/GWPROGS erzeugt File/Directory.	PRODOS
DELETE f,S,D,V Löscht File.	DELETE Filename	DOS PRODOS
EXEC F,R,S,D,V	EXEC PROG.TEXT Führt Textfile aus.	DOS PRODOS
FLUSH p	FLUSH TEXTDAT überträgt Puffer in File.	PRODOS
FP	FP schaltet auf APPLESOFT Basic. löscht Programm und Daten.	DOS
IN#	IN#2 schaltet Input auf Slotnr.	DOS PRODOS

Bezeichnung	Beispiel	Betriebsystem
INIT f	INIT HELLO Formatiert Diskette.	DOS
INT	INT schaltet auf Integer-Basic. löscht Programm und Daten.	DOS
LOAD f,S,D,V	LOAD ROT.MOT Liest File von Diskette.	DOS PRODOS
LOCK f,S,D,V	LOCK ROT.MOT schützt File gegen Schreiben und Löschen.	DOS PRODOS
MAXFILES	MAXFILES 2 reserviert Speicherplatz für Anzahl offener Files.	DOS
MON C,I,O	MON I,O zeigt Befehle und Datenstrom von/zur Diskette.	DOS
NOMON	NOMON neutralisiert MON.	DOS
OPEN f,S,D,V	OPEN SESAM,D2 Öffnet File.	DOS PRODOS
POSITION f,R	POSITION NUM.DAT,R911 ändert Zeiger auf Eintrag -nummer um den angegebenen Betrag.	DOS PRODOS
PR#	PR#3 schaltet Output auf Slotnr.	DOS PRODOS

Bezeichnung	Beispiel	Betriebsystem
PREFIX p	PREFIX/USERS.DISK setzt Prefix.	PRODOS
READ f,S,D,V	READ SESAM markiert File für Input. (bei RANDOM ACCESS Files mit Angabe der Recordnr. READ RANDAC,R33).	DOS PRODOS
RENAME f,g,S,D,V	RENAME HELLO/HALLO ändert Filenamen.	DOS PRODOS
RESTORE p,S,D,V	RESTORE DISK14/NUM.SU liest Programmvariablen von Diskette ein.	PRODOS
RUN f,S,D,V	RUN SUPERFILE,S6,D1,V4 liest und startet Programm.	DOS PRODOS
SAVE f,S,D,V	SAVE ROT.MOT,D2,V13 speichert Programm auf Disk.	DOS PRODOS
STORE p,S,D,V	STORE DISK14/NUM.SU speichert Programmvariable auf Disk.	PRODOS
VERIFY f,S,D,V	VERIFY SUPERFILE prüft Lesbarkeit von File.	DOS PRODOS
WRITE f,S,D,V	WRITE SESAM,D2,V2 markiert File für Output.	DOS PRODOS

Anhang 3

Unterschiede zwischen Apple //c und //e

Der Abschnitt beschreibt interne Unterschiede zwischen den beiden Computern betreffend CPU, Systemprogrammen und Speicheraufteilung.
Der Apple //c kann anhand der Inhalte in den Adressen $FBB3 ($06) und $FBC0 ($00) durch Programme von anderen Apple // Computern unterschieden werden.

CPU

Der Apple //c besitzt den 65C02 8 Bit Mikroprozessor.
Die 65C02 CPU unterscheidet sich von der in älteren Apple Computern verwendeten 6502 im Wesentlichen durch zusätzliche Instruktionen. Die Instruktionen sind im Anhang dargestellt; die auf dem 6502 Prozessor nicht vorhandenen Intruktionen sind dort gekennzeichnet.

Neue Instruktionen

Mn.	hex	Beschreibung
BRA	80	Branch always relative
DEA	3A	Decrement accu
INA	1A	Increment accu
PHX	DA	Push X on Stack
PHY	5A	Push Y on Stack
PLX	FA	Pull X from Stack
PLY	7A	Pull Y from Stack
STZ	9C	Store Zero Absolute
STZ	9E	Store Zero Abs,X
STZ	64	Store Zero Zeropage
STZ	74	Store Zero Zeropage,X
TRB	1C	Test and reset memory bits with accu abs
TRB	14	Test and reset memory bits with accu ZPG
TSB	0C	Test and set memory bits with accu abs
TSB	04	Test and set memory bits with accu ZPG

Neue Adressierungsarten für bestehende Instruktionen

Mn.	hex	Beschreibung
ADC	72	Add memory to accu with carry (ZPG)
AND	32	„AND“ memory with accu (ZPG)
BIT	3C	Test memory bits with accu Abs,X
BIT	34	Test memory bits with accu ZPG,X
CMP	D2	Compare memory with accu (ZPG)
EOR	52	„Exclusive or“ mem. with accu (ZPG)
JMP	7C	Jump Abs(Ind,X)
LDA	B2	Load accu with memory (ZPG)
ORA	12	„OR“ memory with accu (ZPG)
SBC	F2	Subtract memory from accu (ZPG)
STA	92	Store accu in memory (ZPG)

Beide CPUs arbeiten mit der gleichen Taktfrequenz. Ein Takt dauert 0,978 Mikrosekunden. Mit Ausnahme von 7 Instruktionen benötigen beide Prozessoren für die Instruktionen gleiche Taktzahl. Signifikante Geschwindigkeitsunterschiede treten daher nicht auf.

Die auf beiden Prozessoren vorhandenen Instruktionen liefern mit Ausnahme der beiden Folgenden gleiche Resultate:

BIT	immediate	Code $89 verändert bei der 65C02 das Overflow- und das Vorzeichenbit (6 und 7) des Statusregisters nicht.
JMP	indirect	Code $6C holt das höherwertige Adressbyte auf der nächsten Page, wenn als Parameter das letzte Byte einer Page verwendet wurde. (z.B. LSB $1FF, HSB $200) Die 6502 verwendet in diesem Fall das höherwertige Byte von derselben Page (z.B LSB $1FF, HSB $100).

Systemprogramme

Im Apple //c sind zwischen $400 und $7FF zusätzlich zu den Textdaten in den „Screenholes" Informationen über die seriellen Anschlüsse und Mausdaten gespeichert.

Die Hardwarepage ($C000 bis $C100) enthält wesentliche Unterschiede des Apple //c gegenüber dem Apple//e und anderen Apple // Computern. In diesem Bereich befinden sich die Flags zur Memorykontrolle, Bankswitching, Maushandling, serielles Portmanagement, sowie Displaykontrolle und Interrupthandling.
Die Hardwarepage $C000 bis $C100 des Apple //c ist im Anhang dargestellt.

Der Monitor ($F800 bis $FFFF) im Apple //c ist nicht identisch mit dem des Apple //e. Mit einer DOS 3.3 Diskette, die (50 Sektor-) Integer-Basic enthält kann der „alte" Monitor eingelesen werden. Mit dem Befehl INT wird neben Integer-Basic auch der „alte" Monitor aktiviert. Das 42-Sektor INTBASIC enthält keinen Monitor.

Der Apple //c besitzt einen eingebauten Interrupt-Handler im ROM $C803 bis $C89F. Der Interrupt Handler wird bei einem IRQ oder BRK Interrupt über JMP ($FFFE) (ROM) angesprungen, nachdem der Prozessor das laufende Programm abgesichert hat. Die Details der Interrupttechnik sind im Apple //c Reference Manual beschrieben.

Die Mausinterrupts werden durch die Maus-Firmware behandelt. Eigene Interrupt-Routinen können unter Zuhilfenahme bestehender Mausroutinen formuliert werden. Diese Routinen sind im Abschnitt über die Programmierung der Maus dargestellt.

Im Apple //c sind zwei serielle Schnittstellen eingebaut, die durch Software als Ports simuliert werden, um Kompatibilität mit den Serial Cards des Apple //e und früherer Modelle zu gewährleisten. Die Einstellung der Ports ist in Kapitel 5 beschrieben.

Speicheraufteilung

Der Apple //c besitzt 128 kB RAM, aufgeteilt in 2 Banks mit je 64 kB. Davon wird normalerweise nur die 1. Bank (die unteren 64 kB) verwendet.
Diese 1. Bank wird von $0000 bis $FFFF adressiert. Die obersten 12 kB RAM können auf 12 kB ROM mit gleicher Adressierung ($D000 bis $FFFF) umge-

schaltet werden. Zusätzlich sind nochmals 4 kB RAM ($D000 bis $DFFF) als Bank 2 vorhanden.

Der Bereich $C000 bis $FFFF (entpricht der Language Card) enthält:

Hardware Page	$C000 bis $C100
In-Out Firmware	$C100 bis $CFFF
Applesoft Interpreter	$D000 bis $F7FF
Monitor	$F800 bis $FFFF

Die Hardware Page ist im Anhang 10 beschrieben.
Der ROM Bereich $C100 bis $CFFF ist aufgeteilt in:

Serieller Anschluss 1	: Page $C1
Serieller Anschluss 2	: Page $C2
Video Output	: Page $C3
Maus Firmware	: Page $C4
Disk I/O Firmware	: Page $C6
Weitere Video Firmware	
etc.	: Pages $C8 bis $CF

Anhang 4

Fehlermeldungen des Applesoft Basic Interpreters und der Betriebssysteme DOS3.3 und ProDOS. Referenzliste – Fehlercodes

APPLESOFT Basic – Fehlermeldungen

?BAD SUBCRIPT ERROR
Eine dimensionierte Variable wurde nicht mit der in der DIM Anweisung angegebenen Zahl Indices verwendet, oder einer der Indices liegt nicht im zulässigen dimensionierten Wertbereich.

?CAN'T CONTINUE ERROR
Unzulässiger Versuch, ein Programm mit dem Befehl CONT fortzusetzen, z.B. nachdem das Programm geändert wurde.

?DIVISION BY ZERO ERROR
Unzulässige Division durch 0.

?FORMULA TOO COMPLEX ERROR
Mehr als zwei IF Textvergleich THEN ... Anweisungen wurden verwendet.

?ILLEGAL DIRECT ERROR
Unzulässige Verwendung der Befehle DATA, DEF FN, INPUT, GET RESUME, oder ONERR GOTO außerhalb vom Programm.

?ILLEGAL QUANTITY ERROR
Verwendung einer Zahl außerhalb des zulässigen Wertebereichs.

?NEXT WITHOUT FOR ERROR
Verwendung eines NEXT Befehls ohne vorherigen zugehörigen FOR Befehl.

?OUT OF DATA ERROR
Verwendung von mehr READ Anweisungen (ohne RESTORE), als (mit DATA) gespeicherte Zahlen oder Texte vorhanden sind.

?OUT OF MEMORY ERROR
Zuwenig Speicherplatz; zulanges Programm, zuviele Variablen, mehr als 36

geschachtelte Klammern, mehr als 24 geschachtelte Unterprogramme oder mehr als 10 FOR ... NEXT Schleifen.

?OVERFLOW ERROR
Verwendung einer Zahl oder Rechnung mit Ergebnis größer als 1.7E+38 oder absolut kleiner als -1.7E+38.

?REDIM'D ARRAY ERROR
Unzulässige Dimensionierung einer bereits dimensionierten Variable.

?RETURN WITHOUT GOSUB ERROR
Verwendung eines RETURN's ohne vorheriges GOSUB.

?STRING TOO LONG ERROR
Zeichenkette enthält mehr als 255 Zeichen.

?SYNTAX ERROR
Verwendung eines unbekannten oder falsch geschriebenen Befehls oder Fehler ohne besondere eigene Fehlermeldung.

?TYPE MISMATCH ERROR
Verwendung einer Textvariablen für numerischen Ausdruck oder Zahl, oder umgekehrt Verwendung einer numerischen Variable für einen Text.

?UNDEFINED FUNCTION ERROR
Verwendung einer nicht definierten Funktion.

?UNDEFINED STATEMEMT ERROR
Verwendung eines Sprungbefehls (GOTO, GOSUB etc.) mit nicht vorhandener Zeilennummer als Sprungadresse.

DOS3.3 – Fehlermeldungen

DISK FULL
Diskette hat nicht genügend Platz, um Datei abzuspeichern.

END OF DATA
Versuch, über das Fileende hinaus zu lesen.

FILE LOCKED
Gebrauch eines BSAVE, DELETE, RENAME, SAVE oder WRITE Befehls für ein mit LOCK geschütztes File.

FILE NOT FOUND
Verwendung eines nicht existenten Files für den Datenzugriff.

FILE TYPE MISMATCH
Verwendung eines für das angegebene File unzulässigen Befehls. z.B. RUN in Verbindung mit Text- oder Binaryfile.

I/O ERROR
Verwendung einer nicht formatierten oder defekten Diskette oder offenes Laufwerk.

LANGUAGE NOT AVAILABLE
Versuch, einen nicht vorhandenen Interpreter aufzurufen (z.B. INT).

NO BUFFERS AVAILABLE
Verwendung eines Befehls für Diskettenzugriff ohne freie I/O Puffer (Siehe MAXFILES)

NOT DIRECT COMMAND
Verwendung eines Befehls, der nur innerhalb von Programmen vorkommen darf.

PROGRAM TOO LARGE
Versuch, ein Program einzulesen, welches größer ist als der freie Speicherplatz.

RANGE ERROR
Verwendung eines unzulässigen Wertes für eine DOS Anweisung, z.B. ,D3.

SYNTAX ERROR
Fehlerhafte Betriebssystemanweisung.

VOLUME MISMATCH
Diskettenbezeichnung stimmt nicht mit dem angebenen Namen überein.

WRITE PROTECT
Verwendung eines SAVE, BSAVE, WRITE oder STORE Befehls bei einer schreibgeschützten Diskette.

ProDOS – Fehlermeldungen

DIRECTORY FULL
Disketteninhaltverzeichnis enthält bereits 51 Files.

DISK FULL
Kein Platz mehr auf der Diskette.

DUPLICATE FILENAME
Der bei CREATE oder RENAME angegebene neue Filename existiert bereits.

END OF DATA
Versuch, über das Fileende hinaus zu lesen.

FILE BUSY
Das angesprochene File ist bereits geöffnet.

FILE LOCKED
File ist schreibgeschützt oder Directory enthält Files, darf daher nicht gelöscht werden.

FILE NOT OPEN
Zugriffsversuch auf ein nicht geöffnetes File.

FILE(S) STILL OPEN
Das Programm hinterläßt offene Files.

FILE TYPE MISMATCH
Zugriffsbefehl nicht zulässig für das angegebene File.

I/O ERROR
Laufwerkstür ist offen oder Diskette ist nicht formatiert oder defekt.

INVALID OPTION
Unzulässige Parameterangabe

NO BUFFERS AVAILABLE
Kein Speicherplatz für zusätzlichen Pufferbereich vorhanden.

NO DEVICE CONNECTED
Die angesprochene Hardware ist nicht vorhanden.

NOT DIRECT COMMAND
Der Befehl ist nur innerhalb von Programmen zulässig.

PATH NOT FOUND
Ein File mit dem angegebenen Namen (incl. Volume- oder Directoryname) existiert nicht.

PROGRAM TOO LARGE
Zuwenig Speicherplatz für Programm vorhanden.

RANGE ERROR
Angegebener Parameter liegt außerhalb des angegebenen Bereichs.

SYNTAX ERROR
Unzulässige Angaben im Befehl (z.B beim Filenamen)

WRITE PROTECTED
Verwendung eines SAVE, BSAVE, WRITE oder STORE Befehls bei einer schreibgeschützten Diskette.

Referenzliste – Fehlercodes

Fehlercode PEEK(222)	Fehlermeldung	erzeugt durch
0	NEXT WITHOUT FOR	APPLESOFT
1	LANGUAGE NOT AVAILABLE	DOS
2	RANGE ERROR	DOS / PRODOS
3	RANGE ERROR	DOS
	NO DEVICE CONNECTED	PRODOS
4	WRITE PROTECTED	DOS / PRODOS
5	END OF DATA	DOS / PRODOS
6	FILE NOT FOUND	DOS
	PATH NOT FOUND	PRODOS
7	VOLUME MISMATCH	DOS
	PATH NOT FOUND	PRODOS
8	I/O ERROR	DOS / PRODOS
9	DISK FULL	DOS / PRODOS
10	FILE LOCKED	DOS / PRODOS
11	SYNTAX ERROR	DOS
	INVALID OPTION	PRODOS
12	NO BUFFERS AVAILABLE	DOS / PRODOS
13	FILE TYPE MISMATCH	DOS / PRODOS
14	PROGRAM TOO LARGE	DOS / PRODOS
15	NOT DIRECT COMMAND	DOS / PRODOS
16	SYNTAX ERROR	APPLESOFT
	SYNTAX ERROR	PRODOS
17	DIRECTORY FULL	PRODOS
18	FILE NOT OPEN	PRODOS
19	DUPLICATE FILENAME	PRODOS
20	FILE BUSY	PRODOS
21	FILE(S) STILL OPEN	PRODOS

Fehlercode PEEK(222)	Fehlermeldung	erzeugt durch
22	RETURN WITHOUT GOSUB	APPLESOFT
42	OUT OF DATA	APPLESOFT
53	ILLEGAL QUANTITY	APPLESOFT
69	OVERFLOW	APPLESOFT
77	OUT OF MEMORY	APPLESOFT
90	UNDEFINED STATEMENT	APPLESOFT
107	BAD SUBSCRIPT	APPLESOFT
120	REDIMENSIONED ARRAY	APPLESOFT
133	DIVISION BY ZERO	APPLESOFT
163	TYPE MISMATCH	APPLESOFT
176	STRING TOO LONG	APPLESOFT
191	FORMULA TOO COMPLEX	APPLESOFT
224	UNDEFINED FUNCTION	APPLESOFT
254	Fehler bei INPUT	APPLESOFT
255	Eingabe von CONTROL-C	APPLESOFT

Anhang 5

Reservierte Worte

ABS
AND
APPEND
ASC
AT
ATN
BLOAD
BRUN
BSAVE
CALL
CAT
CATALOG
CHAIN
CHR$
CLEAR
CLOSE
COLOR=
CONT
COS
CREATE
DATA
DEF
DEL
DELETE
DIM
DRAW
END
EXEC
EXP
FLASH
FLUSH
FN
FOR
FP
FRE
GET
GOSUB
GOTO
GR
HCOLOR=
HGR
HGR2
HIMEM:
HLIN
HOME
HPLOT
HTAB
IF
IN#
INIT
INPUT
INT
INVERSE
LEFT$
LEN
LET
LIST
LOAD
LOCK
LOG
LOMEM:
MAXFILES
MID$
MON
NEW
NEXT
NOMON
NORMAL
NOT
NOTRACE
ON
ONERR
OPEN
OR
PDL
PEEK
PLOT
POKE
POP
POS
POSITION
PR#
PREFIX
PRINT
READ
RECALL
REM
RENAME
RESTORE
RESUME
RETURN
RIGHT$
RND
ROT=
RUN
SAVE
SCALE=
SCRN(
SGN
SHLOAD
SIN
SPC(
SPEED=
SQR
STEP

STOP	THEN	VERIFY
STORE	TO	VLIN
STR$	TRACE	VTAB
TAB(	UNLOCK	WAIT
TAN	USR	WRITE
TEXT	VAL	XDRAW

Anhang 6

Speicheraufteilung

Die Aufteilung des RAM Speichers des Apple //c hängt vom verwendeten Betriebssystem (DOS 3.3 oder ProDOS) ab. In den Darstellungen wird die erste 64 kB Bank beschrieben, die Bank 2 ist normalerweise ungenutzt. Die Verwendung der Bank 2 als RAM-Disk ist in Kapitel 3 beschrieben.

Speicheraufteilung nach Boot mit DOS 3.3

Adresse dez	Adresse hex	Art	belegt durch
0- 255 (Zeropage)	0-FF	RAM	Systemvariable
256-511	100-1FF	RAM	Stapel/Stack
512-767	200-2FF	RAM	Tastaturpuffer
768-975	300-3CF	RAM	normalerweise frei
976-1023	3D0-3FF	RAM	Monitorvektoren
1024-2047	400-7FF	RAM	Bildschirmdaten (TEXT und GR)
2048-8191	800-1FFF	RAM	frei für BASIC Programme + Daten
8192-16383	2000-3FFF	RAM	Grafik (HGR) oder Programme + Daten
16384-24575	4000-5FFF	RAM	Grafik (HGR2) oder Programme + Daten

Adresse dez	Adresse hex	Art	belegt durch
24575-38399	6000-95FF	RAM	frei für BASIC Daten (ev. Programme)
38400-49151	9600-BFFF	RAM	DOS3.3
49152-49407 (Hardwarepage)	C000-C0FF	RAM	I/O Vektoren
49408-65536	C100-FFFF	ROM	Applesoft Interpreter Monitor usw.

Speicheraufteilung mit ProDOS

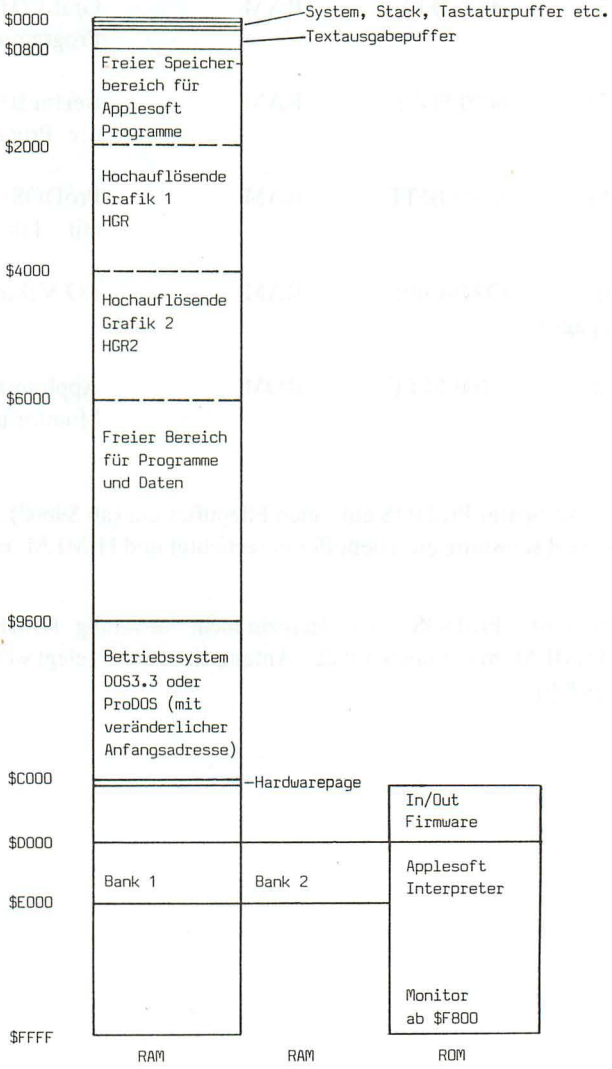
Adresse dez	Adresse hex	Art	belegt durch
255 (Zeropage)	0-FF	RAM	Systemvariable
256-511	100-1FF	RAM	Stapel/Stack
512-767	200-2FF	RAM	Tastaturpuffer
768-975	300-3CF	RAM	normalerweise frei
976-1023	3D0-3FF	RAM	Monitorvektoren
1024-2047	400-7FF	RAM	Bildschirmdaten (TEXT und GR)
2048-8191	800-1FFF	RAM	frei für BASIC Programme + Daten
8192-16383	2000-3FFF	RAM	Grafik (HGR) oder Programme + Daten

Adresse dez	Adresse hex	Art	belegt durch
16384-24575	4000-5FFF	RAM	Grafik (HGR2) oder Programme + Daten
24575-36351	6000-8DFF	RAM	frei für BASIC Daten (ev. Programme)
36352-49151	8E00-BFFF	RAM	ProDOS mit 3 Filepuffern
49152-49407 (Hardwarepage)	C000-C0FF	RAM	I/O Vektoren
49408-65536	C100-FFFF	ROM	Applesoft Interpreter Monitor usw.

Normalerweise richtet ProDOS nur einen Filepuffer ein (ab $9600). Für jedes offene File wird selbstätig ein Filepuffer eingerichtet und HIMEM: nach unten verschoben.

Anmerkung: Unter ProDOS ist es allgemein nicht notwendig, HIMEM: selbst zu setzen; HIMEM: muß immer auf den Anfang einer Seite gelegt werden (z.B. HIMEM: $9200).

Speicheraufteilung der ersten 64 kB Speicherbereich.



Anhang 7

ASCII-Code Tabelle

ASCII = American Standard Code for Information Interchange

Der Apple //c verfügt über zwei Zeichensätze, genannt primärer und alternativer Zeichensatz.

Der primäre Zeichensatz wird durch PR#0 erreicht.

Der alternative Zeichensatz wird durch Esc-4, Esc-8 oder PR#3 aktiviert.

Ist der alternative Zeichensatz aktiviert, so werden durch PRINT CHR$(27) anstelle der invers dargestellten Grossbuchstaben die Mauszeichen ausgegeben.
Nach PRINT CHR$(24) werden beim alternativen Zeichensatz wieder die inversen Grossbuchstaben dargestellt.

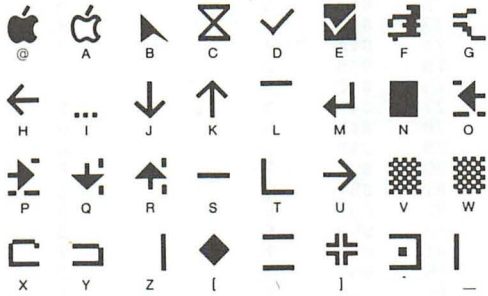
Die Mauszeichen sind unten dargestellt.

Durch den rechten Schalter oberhalb der Tastatur wird zwischen US - und deutschem Zeichensatz gewählt. Die Zeichensätze entsprechen den in Kapitel 1 dargestellten Tastaturaufteilungen.

ASCII dez	CODE hex	Bedeutung	Taste
0	$00		CONTROL-@
1	$01		CONTROL-A
2	$02		CONTROL-B
3	$03		CONTROL-C
4	$04		CONTROL-D
5	$05		CONTROL-E
6	$06		CONTROL-F
7	$07	Bell/Piepston	CONTROL-G
8	$08	Backspace	CONTROL-H
9	$09		CONTROL-I
10	$0A	Zeilenvorschub	CONTROL-J
11	$0B		CONTROL-K
12	$0C		CONTROL-L
13	$0D	Wagenruecklauf	CONTROL-M
14	$0E		CONTROL-N
15	$0F		CONTROL-O
16	$10		CONTROL-P
17	$11		CONTROL-Q
18	$12		CONTROL-R
19	$13		CONTROL-S
20	$14		CONTROL-T
21	$15	Tabulator	CONTROL-U
22	$16		CONTROL-V
23	$17		CONTROL-W
24	$18	Zeile loeschen	CONTROL-X
25	$19		CONTROL-Y
26	$1A		CONTROL-Z
27	$1B		esc
28	$1C		
29	$1D		CONTROL-SHIFT-M
30	$1E		CONTROL-^
31	$1F		
32	$20	Leerzeichen	Leertaste
33	$21	!	!
34	$22	"	"
35	$23	#	#
36	$24	$	$
37	$25	%	%
38	$26	&	&
39	$27	'	'
40	$28	(	(
41	$29	)	)
42	$2A	*	*
43	$2B	+	+
44	$2C	,	,
45	$2D	-	-
46	$2E	.	.
47	$2F	/	/
48	$30	0	0
49	$31	1	1
50	$32	2	2
51	$33	3	3
52	$34	4	4
53	$35	5	5
54	$36	6	6
55	$37	7	7
56	$38	8	8

57	$39	9	9
58	$3A	:	:
59	$3B	;	;
60	$3C	<	<
61	$3D	=	=
62	$3E	>	>
63	$3F	?	?
64	$40	@	@
65	$41	A	A
66	$42	B	B
67	$43	C	C
68	$44	D	D
69	$45	E	E
70	$46	F	F
71	$47	G	G
72	$48	H	H
73	$49	I	I
74	$4A	J	J
75	$4B	K	K
76	$4C	L	L
77	$4D	M	M
78	$4E	N	N
79	$4F	O	O
80	$50	P	P
81	$51	Q	Q
82	$52	R	R
83	$53	S	S
84	$54	T	T
85	$55	U	U
86	$56	V	V
87	$57	W	W
88	$58	X	X
89	$59	Y	Y
90	$5A	Z	Z
91	$5B	[	[
92	$5C	\	\
93	$5D	]	]
94	$5E	^	^
95	$5F		
96	$60	`	`
97	$61	a	a
98	$62	b	b
99	$63	c	c
100	$64	d	d
101	$65	e	e
102	$66	f	f
103	$67	g	g
104	$68	h	h
105	$69	i	i
106	$6A	j	j
107	$6B	k	k
108	$6C	l	l
109	$6D	m	m
110	$6E	n	n
111	$6F	o	o
112	$70	p	p
113	$71	q	q
114	$72	r	r
115	$73	s	s
116	$74	t	t

117	$75	u	u
118	$76	v	v
119	$77	w	w
120	$78	x	x
121	$79	y	y
122	$7A	z	z
123	$7B	{	{
124	$7C	¦	¦
125	$7D	}	}
126	$7E	~	~
127	$7F		

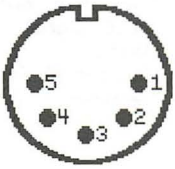


Anhang 8

Pinbelegung der Anschlüsse

Stromversorgung

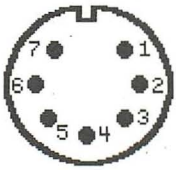
Serielle Anschluesse 1 und 2



Pin	Signal
1	Data Terminal Ready output
2	Transmit Data output
3	Power and signal common
4	Read Data input
5	Data Set Ready input

Serielle Anschlüsse 1 und 2

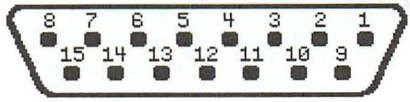
Stromversorgung



Pin	Signal
1	Nicht angeschlossen
2,3	Masse
4	Gehaeuseerdung
5,6	15 V Gleichspannung
7	Nicht angeschlossen

Anschluss für 2.Diskettenlaufwerk

15 Pol Videoausgang

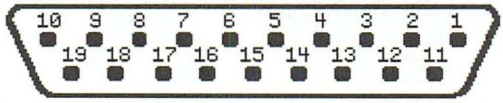


Pins	Bedeutung	Pins	Bedeutung
1	Textsignal	9	RAM la strobe
2	14 MHz Signal	10	Graphics en.
3	Vert.+hor. Synchr.	11	Ser. char. outp.
4	Vert. Zaehlbit	12	NTSC Signal
5	Soundsignal	13	Masse
6	Video shift reg. en.	14	Halbdot shift
7	Aktiver Bereich	15	Farbref. Signal
8	+ 12 V		

Video

Videoausgang

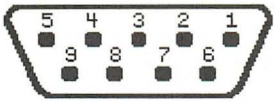
Anschluss fuer 2. Disk



Pins	Bedeutung	Pins	Bedeutung
1,2,3,4	Masse	15	Write Request
5	-12 V	16	Drive 1 select
6	+5 V	17	Read data
7,8	+12 V	18	Write data
9	External Interrupt		
10	Write-protect input		
11-14	Motor-phase 0-3 output		

Mausanschluß

Mausanschluss



Pin	Bedeutung	
1	Mausindentifier	(auch Button1)
2	Stromversorgung 5V	
3	Systemmasse	
4	X1, Eingang fuer X Richtung	
5	X0, X-Bewegungsinterrupt	(auch PDL0)
6	Maustaste (Button)	
7	Maustaste (Button)	(auch Button0)
8	Y1, Eingang fuer Y Richtung	(auch PDL1)
9	Y0, Y-Bewegungsinterrupt	

Anhang 9

Maschinenprogramme und Adressen

Die Liste erhebt keinen Anspruch auf Vollständigkeit.

Sind zwei benachbarte Adressen (z.B. 103,104 LL,HH) angegeben, so wird der Gesamtwert ermittelt durch:
PRINT PEEK(LL) + 256 * PEEK (HH)

Verzeichnis nach Bereichen

Bereich	Adresse dezimal	Adresse hex	Aufruf
Adresse	103,104	67,68	PEEK
	Beginn des BASIC Programms		
Adresse	105,106	69,6A	PEEK
	LOMEM:, Beginn der Variablen		
Adresse	107,108	6B,6C	PEEK
	Ende Variablen / Anfang dim. Variablen		
Adresse	109,110	6D,6E	PEEK
	Ende dim. Variablen / Anfang Stringpool		
Adresse	111,112	6F,70	PEEK
	Momemtanes unteres Ende für Strings		
Adresse	115,116	73,74	PEEK
	HIMEM: / Obergrenze Strings		
Adresse	43626,43627		PEEK
	Länge Maschinenprogramm nach BLOAD		
Adresse	43634,43635		PEEK
	Anfang Maschinenprogramm nach BLOAD		

Bereich	Adresse dezimal	Adresse hex	Aufruf
Akustik	-1059 Piepston erzeugen	FBDD	CALL
Akustik	-1052 1 kHz Piepston erzeugen	FBE4	CALL
Akustik	-198 BELL-Zeichen (ASCII 7) ausgeben	FF3A	CALL
Ausgabe	-626 CR ausgeben	FD8E	CALL
Ausgabe	-531 Zeichen ausgeben, ASCII Wert in A	FDED	PRINT
Ausgabe	-528 Inhalt von A darstellen, ASCII in A	FDF0	PRINT
Datum/ProDOS	49040 Tag + Monat * 32 (Monat < 8)		POKE
Datum/ProDOS	49040 Tag + (Monat-8) * 32 (wenn Monat > 7)		POKE
Datum/ProDOS	49041 Jahr x 2 + 1 (wenn Monat > 7)		POKE
Display	-1720 3 Leerzeichen ausgeben	F948	CALL
Display	-1718 n Leerzeichen ausgeben, Anzahl n in A	F94A	CALL
Display	-1233 Bildschirm auf Text umschalten	FB2F	TEXT

Bereich	Adresse dezimal	Adresse hex	Aufruf
Display	-1184	FB60	CALL
	Bildschirm löschen, Ausgabe: Apple //c		
Display	-1008	FC10	CALL
	Cursor nach links		
Display	-998	FC1A	CALL
	Cursor nach oben		
Display	-958	FC42	CALL
	Bildschirm ab Cursor nach unten löschen		
Display	-954	FC46	CALL
	Bildschirm ab Y,A nach unten löschen Spalte in Y, Zeile in A		
Display	-936	FC58	HOME
	Bildschirm löschen, Cursor oben links		
Display	-922	FC66	CALL
	Cursor nach unten		
Display	-912	FC70	CALL
	Bildschirmtext eine Zeile nach oben		
Display	-868	FC9C	CALL
	Zeile rechts vom Cursor löschen		
Display	-384	FE80	INVERSE
	Inverse Darstellung		
Display	-380	FE84	NORMAL
	Normale Darstellung		
Display	32	0020	POKE 32,LR
	Linken Bildschirmrand setzen		

Bereich	Adresse dezimal	Adresse hex	Aufruf
Display	33 Bildschirmbreite	0021	POKE 33,BR.
Display	34 Oberen Bildschirmrand setzen	0022	POKE 34,OR
Display	35 Unteren Bildschirmrand setzen	0023	POKE 35,UR
Display	50 Invertieren	32	POKE 50,63
Display	50 Normale Darstellung	32	POKE 50,255
DOS/ProDOS	-1370 Bootet von Disk	FAA6	CALL
DOS3.3	-23186 Inhaltverzeichnis des Disk	A56E	CATALOG
DOS3.3	43624 Nummer des aktuellen Disklaufwerks	AA68	PEEK
DOS3.3	43626 Nummer des aktuellen Diskslots	AA6A	PEEK
DOS3.3	44033 Spurnummer für CATALOG	AC01	PEEK
DOS3.3	44567 Max. Länge Filenamen für CATALOG	AE17	POKE44567,Z
DOS3.3	44605 CATALOG Zeilenzahl bis zur Pause	AE3D	PEEK

Bereich	Adresse dezimal	Adresse hex	Aufruf
DOS3.3	44617,44618	AE49,AE4A	POKE
	CATALOG ohne Blocklänge POKE44617,234 POKE44618,234		
DOS3.3	45620,45621	B234,B235	POKE
	CATALOG nur mit 1.File POKE45620,234 POKE45621,234		
Drucker	-926	FC62	CALL
	Wagenrücklauf und Zeilenvorschub ausgeben		
Eingabe	-665	FD67	INPUT
	CR, dann Zeichen einlesen, Hinweis in 51, Stringlänge in X, Zeichen ab $200		
Eingabe	-662	FD6A	CALL
	Zeichen einlesen (mit Cursor)		
Eingabe	-657	FD6F	CALL
	Zeichen einlesen (ohne Cursor)		
Erkennung	-1101	FBB3	PEEK
	Maschinentyp prüfen (A//c) Apple//c = 06		
Erkennung	-1088	FBC0	PEEK
	Maschinentyp prüfen (A//c) Apple//c = 00		
Fehler	-211	FF2D	CALL
	Piepston und ERR ausgeben		
Grafik	62454	F3F6	CALL
	Löscht Hires Grafik		
Interpreter	-7036	E484	CALL
	Unnötige Strings entfernen, Garbage Collection		

Bereich	Adresse dezimal	Adresse hex	Aufruf
Interpreter	-756	FD0C	CALL
	Stoppt Programm, erwartet Tasteneingabe		
Interpreter	-678	FD5A	CALL
	Stoppt Programm, erwartet RETURN		
Interpreter	-676	FD5C	CALL
	Stoppt Programm, piepst, erwartet RETURN		
Interpreter	-336	FEB0	CALL
	Programm und Daten löschen, Interpreter aufrufen		
Interpreter	214	D6	POKE 214,255
	Jeder Befehl startet BASIC Programm		
Interpreter	1012	3F4	POKE 1012,0
	Control-Reset bootet von Disk		
Interpreter	1014,1015	3F6,3F7	PEEK
	Zeiger für Sprung bei & Zeichen		
Interpreter	2049	801	POKE 2049,1
	LIST Befehl auf erste Programmzeile begrenzen		
Interpreter	2050	802	POKE 2050,0
	APPLESOFT Speicherbereich löschen		
Lores Gr	-2048	F800	PLOT
	Bildpunkt erzeugen Spalte in Y, Zeile in A		
Lores Gr	-2023	F819	HLIN
	horizontale Linie erzeugen L. Spalte in Y, R. Spalte in 44, Zeile in A		

Bereich	Adresse dezimal	Adresse hex	Aufruf
Lores Gr	-2008	F828	VLIN
	Verikale Linie erzeugen Spalte in Y, O. Zeile in A, U. Zeile in 45		
Lores Gr	-1998	F832	CALL
	Löschen		
Lores Gr	-1994	F836	GR
	Obere 40 Zeilen löschen		
Lores Gr	-1953	F85F	CALL
	COLOR= Wert um 3 erhöhen		
Lores Gr	-1948	F864	COLOR=
	Farbe auswählen, Farbcode in A		
Lores Gr	-1935	F871	SCRN
	Farbe eines Punktes ermitteln Spalte in Y, Zeile in A, Farbe in A		
Lores Gr	-1216	FB40	GR
	40 Zeilengrafik ein		
Mauseingang	-1250	FB1E	PDL( )
	Eingabewert von Anschluss lesen Paddlenummer in X, (0 bis 3) Wert in Y (0 bis 255)		
Monitor	-468	FE2C	CALL
	Memory kopieren (Move)		
Monitor	-458	FE36	CALL
	Speicher vergleichen (Verify)		
Monitor	-151	FF69	CALL
	Monitor aufrufen		

Bereich	Adresse dezimal	Adresse hex	Aufruf
Register	-1728	F940	CALL
	Inhalt von X und Y als XXYY hex ausgeben		
Register	-1727	F941	CALL
	Inhalt von A und X als AAXX hex ausgeben		
Register	-1724	F944	CALL
	Inhalt von X als XX hex ausgeben		
Register	-550	FDDA	CALL
	Inhalt von A als AA hex ausgeben		
Register	-193	FF3F	CALL
	Gespeicherte Register einsetzen A-Wert in 45, X-Wert in 46 Y-Wert in 47		
Register	-182	FF4A	CALL
	Reisterinhalte speichern A-Wert in 45, X-Wert in 46 Y-Wert in 47		
System	-1691	FA47	CALL
	Break handler		
System	-1438	FA62	CALL
	Reset Routine		
System	-856	FCA8	CALL
	Verzögerungroutine Anzahl Durchläufe in A		
Tastatur	-756	FD1B	CALL
	Eingabe abwarten, außer ESC ASCII Wert in A		

Bereich	Adresse dezimal	Adresse hex	Aufruf
Tastatur	-715 Eingabe abwarten ASCII Wert in A	FD35	CALL

Anhang 10

Hardwarepage ($C000 bis $C100)

Die Spalte Aufruf bedeutet

R =	READ, entweder PEEK(Adresse) oder LDA Adresse
W =	WRITE, entweder POKE Adresse,Wert oder STA Adresse
R7 =	READ Bit 7 (Prüfung ob Bit 7=1, bzw. Wert > 127) IF PEEK(Adresse) > 127 THEN
R/W=	READ oder WRITE

Typ	Funktion	Aufruf	Adresse dez	Adresse hex
Tastatur	Prüfen ob Taste gedrückt Taste gedrückt wenn PEEK größer als 127	R7	49152	C000
Display	Page 2 auf Aux RAM	W	49152	C000
Display	Page 2 auf Main RAM	W	49153	C001
Memory	Read Main 48K	W	49154	C002
Memory	Read aux 48K	W	49155	C003
Memory	Write in Main 48K	W	49156	C004
Memory	Write in aux 48K	W	49157	C005
Memory	Main P0,P1	W	49160	C008
Memory	Aux P0,P1	W	49161	C009

Typ	Funktion	Aufruf	Adresse dez	Adresse hex
Display	40 Zeichen /Zeile Text	W	49164	C00C
Display	80 Zeichen /Zeile Text	W	49165	C00D
Display	Normaler Zeichensatz ein	W	49166	C00E
Display	Alt. Zeichensatz ein	W	49167	C00F
Tastatur	Strobe löschen Immer nach PEEK(-16336) durchführen	W	49168	C010
Memory	Read D000 Bank2/1 switch	R7	49169	C011
Memory	Read RAM/ROM switch	R7	49170	C012
Memory	Read Main/Aux 48K Read switch	R7	49171	C013
Memory	Read Main/Aux 48K Write switch	R7	49172	C014
Maus	Reset Maus X interrupt	R	49173	C015
Memory	Read Main/Aux P0,P1 switch	R	49175	C016
Maus	Reset Maus Y interrupt	R	49175	C017
Display	Read Page 2 Main/Aux switch	R7	49176	C018
Grafik	Read GR/Text Mixed switch	R7	49179	C01B
Display	Read Page 1/2 switch	R7	49180	C01C
Grafik	Read Hires/Lores switch >127 wenn Hires	R7	49181	C01D

Typ	Funktion	Aufruf	Adresse dez	Adresse hex
Display	Read Norm/Alt Zeichensatz switch	R7	49182	C01E
Display	Read 40/80 switch	R7	49183	C01F
Grafik	Read Text/Grafik switch	R7	49187	C01A
Akustik	Lautsprecher klicken	W	49200	C030
Maus	löscht Maus Interrupt	R	49224	C048
Grafik	Grafik einschalten	R/W	49232	C050
Display	Text einschalten	R/W	49233	C051
Grafik	nur Grafik /nur Text	R/W	49234	C052
Grafik	Grafik mit Text	R/W	49235	C053
Display	Page 1 ein	R/W	49236	C054
Display	Page 1X oder 2 ein	R/W	49237	C055
Grafik	Lores Grafik einschalten	R/W	49238	C056
Grafik	Hires Grafik einschalten	R/W	49239	C057
Maus	Maus Interrupts aus	R	49240	C058
Maus	Maus Interrupts ein	R	49241	C059
Grafik	DHires ein (wenn IOU ein)	R/W	49246	C05E
Grafik	DHires aus (wenn IOU ein)	R/W	49247	C05F
Display	Read 80/40 switch	R7	49248	C060

Typ	Funktion	Aufruf	Adresse dez	Adresse hex
Tastatur	0 oder OPEN-APPLE gedrueckt?	R7	49249	C061
Tastatur	1 oder CLOSED-APPLE gedrueckt?	R7	49250	C062
Maus	Read Maus switch/button	R7	49251	C063
Maus	Maus X Richtung lesen	R	49254	C066
Maus	Maus Y Richtung lesen	R	49255	C067
Maus	Trigger Analog Eingang	R	49264	C070
Grafik	READ IOU switch	R7	49278	C07E
Grafik	IOU ein (DHGR möglich)	W	49278	C07E
Grafik	IOU aus	W	49279	C07F
Grafik	Read Dhires switch	R7	49279	C07F
Memory	Read RAM, D000 Bank 2	R	49280	C080
Memory	Read ROM, write RAM, D000 Bank 2	RR	49281	C081
Memory	Read ROM, D000 Bank 2	R	49283	C083
Memory	Read and Write RAM, D000 Bank 2	RR	49283	C083
Memory	Read RAM, D000 Bank 1	R	49288	C088
Memory	Read ROM, write RAM, D000 Bank 1	RR	49289	C089
Memory	Read ROM, D000 Bank 1	R	49290	C08A
Memory	Read and Write RAM, D000 Bank 1	RR	49291	C08B

Typ	Funktion	Aufruf	Adresse dez	Adresse hex
Port	1 ACIA Transmit/receive Register	R/W	49304	C098
Port 1	ACIA Status Register	R/W	49305	C099
Port 1	ACIA Command register	R/W	49306	C09A
Port 1	ACIA Control Register	R/W	49307	C09B
Port 2	ACIA Transmit/receive Register	R/W	49320	C0A8
Port 2	ACIA Status Register	R/W	49321	C0A9
Port 2	ACIA Command Register	R/W	49322	C0AA
Port 2	ACIA Control Register	R/W	49323	C0AB

Anhang 11

Maschinenbefehle des 6502 und des 65C02 Prozessors

(Sonderbefehle des 65C02 sind gekennzeichnet)

Adressierungsarten

		z. Beispiel
IMP	Implizit	DEX
ACCU:	Akkumulator	ASL
IMM:	Unmittelbare Adr.	CPX #$FF
ABS:	Absolute Adr.	LDA $FFFF
ZPG:	Zeropage Adr.	LDA $CE
ABS,X ABS,Y:	Indizierte absolute Adr.	LDA $1000,X
ZPG,Y ZPG,X:	Indizierte Zeropage Adr.	LDA $CE,X
REL:	Relative Adr.	BRA $0303
(IND,X):	Indirekte, indizierte Zeropage Adr.	LDA ($EE,X)
ABS (IND,X)	Absolute, indirekte, indizierte Adr.	JMP ($1000,X)
(IND),Y:	Indirekte, indizierte Adr.	LDA ($EE),Y
(ZPG):	Indirekte Zeropage Adr.	LDA ($EE)
(ABS):	Indirekte Absolute Adr.	JMP ($1000)

Alphabetisch geordnet

ADC Add Memory to Accumulator with Carry

Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu
ADDIERT	IMM	69	2	2	
SPEICHERINHALT	ABS	6D	4	3	
ZUM AKKU	ZPG	65	3	2	
ÄNDERT CARRY	(IND,X)	61	6	2	
	(IND),Y	71	5	2	
	ZPG,X	75	4	2	
	ABS,X	7D	4	3	
	ABS,Y	79	4	3	
	(ZPG)	72	5	2	65C02

AND „AND“ Memory with Accumulator

Log. UND	IMM.	29	2	2	
SPEICHER	ABS	2D	4	3	
UND AKKU	ZPG	25	3	2	
	(IND,X)	21	6	2	
	(IND),Y	31	5	2	
	ZPG,X	35	4	2	
	ABS,X	3D	4	3	
	ABS,Y	39	4	3	
	(ZPG)	32	5	2	65C02

ASL Arithmetic Shift One Bit Left

AKKU ODER SPEICHER	Akku	0A	2	1	
UM EIN BIT	ABSOLUT	0E	6	3	
NACH LINKS	ZPG	06	5	2	
VERSCHIEBEN	ZPG,X	16	6	2	
	ABS,X	1E	6	3	

Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu
BCC Branch on Carry Clear					
VERZWEIGUNG BEI CARRY=0	RELATIV	90	2	2	
BCS Branch on Carry Set					
VERZWEIGUNG BEI CARRY=1	RELATIV	B0	2	2	
BEQ Branch on Result Zero					
VERZWEIGUNG BEI NULL	Relativ	F0	2	2	
BIT Test Memory Bits with Accumulator					
VERGLEICHT	ABS	2C	4	3	
BITS VON	ZPG	24	3	2	
SPEICHER	IMM	89	2	2	
UND AKKU	ZPG,X	34	4	2	65C02
	ABS,X	3C	4	3	65C02
BMI Branch on Result Minus					
VERZWEIGT BEI MINUS	RELATIV	30	2	2	
BNE Branch on Result not Zero (not equal)					
VERZWEIGT BEI >< 0	RELATIV	D0	2	2	

Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu
BPL Branch on Result Plus					
VERZWEIGT BEI PLUS	RELATIV	10	2	2	
BRA Branch Always					
UNBED. VERZWEIGUNG	RELATIV	80	2	2	65C02
BRK Force Break					
INTERRUPT	IMPLIZIT	00	7	1	
BVC Branch on Overflow Clear					
VERZWEIGT BEI V=0	RELATIV	50	2	2	
BVS Branch on Overflow Set					
VERZWEIGT BEI V=1	REL	70	2	2	
CLC Clear Carry Flag					
LÖSCHT C	IMPLIZIT	18	2	1	
CLD Clear Decimal Mode					
LÖSCHT DEZIMAL	IMPLIZIT	D8	2	1	

Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu
CLI Clear Interrupt Disable Flag					
LÖSCHT INTERRUPTSPERRE	IMPLIZIT	58	2	1	
CLV Clear Overflow Flag					
LÖSCHT V	IMPLIZIT	B8	2	1	
CMP Compare Memory and Accumulator					
VERGLEICHE AKKU UND SPEICHER	IMM	C9	2	2	
	ZPG	C5	3	2	
	ABS	CD	4	3	
	(IND,X)	C1	6	2	
	(IND),Y	D1	5	2	
	ZPG,X	D5	4	2	
	ABS,X	DD	4	3	
	ABS,Y	D9	4	3	
	(ZPG)	D2	5	2	65C02
CPX Compare Memory and Index X					
VERGLEICHT REG. X UND SPEICHER	ABS	EC	4	3	
	IMM	E0	2	2	
	ZPG	E4	3	2	
CPY Compare Memory and Index Y					
VERGLEICHT REG. X UND SPEICHER	ABS	CC	4	3	
	IMM	C0	2	2	
	ZPG	C4	3	2	

Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu
DEA Decrement Accumulator					
REDUZIERT AKKU UM 1	AKKU	3A	2	1	65C02
DEC Decrement by One					
REDUZIERT	ABS	CE	6	3	
UM EINS	ZPG	C6	5	2	
	ZPG,X	D6	6	2	
ABS,X	DE	7	3		
DEX Decrement Index X by One					
REZUZIERT X UM EINS	IMPLIZIT	CA	2	1	
DEY Decrement Index Y by One					
REDUZIERT Y UM EINS	IMPLIZIT	88	2	1	
EOR „Exclusive-or“ Memory with Accumulator					
LOGISCHES	IMM	49	2	2	
EXKLUSIVES	ABS	4D	4	3	
ODER,	ZPG	45	3	2	
SPEICHER	(IND,X)	41	6	2	
UND AKKU	(IND),Y	51	5	2	
	ZPG,X	55	4	2	
	ABS,X	5D	4	3	
	ABS,Y	59	4	3	
	(ZPG)	52	5	2	65C02

Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu
INA Increment Accumulator					
ADD. 1 ZU AKKU	AKKU	1A	2	1	65C02
INC Increment by One					
ADDIERE 1	ABS	EE	6	3	
	ZPG	E6	5	2	
	ZPG,X	F6	6	2	
	ABS,X	FE	6	3	
INX Increment Index X by One					
ADD. 1 ZU X	IMPLIZIT	E8	2	1	
INY Increment Index Y by One					
ADD. 1 ZU Y	IMPLIZIT	C8	2	1	
JMP Jump to New Location					
UNBEDINGTER SPRUNG	ABS	4C	3	3	
	IND	6C	6	3	
	(IND,X)	7C	6	3	65C02
JSR Jump to New Location Saving Return Adress					
SPRUNG IN EINE SUBROUTINE	ABS	20	6	3	

Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu
LDA Load Accumulator with Memory					
LIEST	IMM	A9	2	2	
SPEICHER	ABS	AD	4	3	
IN DEN AKKU	ZPG	A5	3	2	
	(IND,X)	A1	6	2	
	(IND),Y	B1	5	2	
	ZPG,X	B5	4	2	
	ABS,X	BD	4	3	
	ABS,Y	B9	4	3	
	IND	B2	5	2	65C02
LDX Load Index X with Memory					
LIEST	IMM	A2	2	2	
SPEICHER	ABS	Ä	4	3	
IN X REG.	ZPG	A6	3	2	
	ZPG,Y	B6	4	2	
	ABS,Y	BE	4	3	
LDY Load Index Y with Memory					
LIEST	IMM	A0	2	2	
SPEICHER	ABS	AC	4	3	
IN Y REG.	ZPG	A4	3	2	
	ZPG,X	B4	4	4	
	ABS,X	BC	4	3	

Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu
LSR Shift One Bit Right					
AKKU	ABS	4E	6	3	
ODER SPEICHER	ZPG	46	5	2	
UM EIN BIT	ABS	4A	2	1	
NACH RECHTS	ZPG,X	56	6	2	
SCHIEBEN	ABS,X	5E	6	3	
NOP No Operation					
KEINE OPERATION	IMPLIZIT	EA	2	1	
ORA „OR" Memory with Accumulator					
LOGISCHES	IMM	09	2	2	
ODER	ABS	0D	4	3	
(SPEICHER	ZPG	05	3	2	
UND AKKU)	(IND,X)	01	6	2	
	(IND),Y	11	5	2	
	ZPG,X	15	4	2	
	ABS,X	1D	4	3	
	ABS,Y	19	4	3	
	IND	12	5	2	65C02
PHA Push Accumulator on Stack					
AKKU ZUM STACK	IMPLIZIT	48	3	1	
PHP Push Processor Status on Stack					
PROZESSOR STATUS ZUM STACK	IMPLIZIT	08	3	1	

Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu
PHX Push Index X on Stack					
X REGISTER ZUM STACK	IMPLIZIT	DA	3	1	65C02
PHY Push Index Y on Stack					
Y REGISTER ZUM STACK	IMPLIZIT	5A	3	1	65C02
PLA Pull Accumulator from Stack					
HOLE AKKU AUS STACK	IMPLIZIT	68	4	1	
PLP Pull Processor Status from Stack					
HOLE PROZ.STATUS AUS STACK	IMPLIZIT	28	4	1	
PLX Pull Index X from Stack					
HOLE X AUS STACK	IMPLIZIT	FA	4	1	65C02
PLY Pull Index Y from Stack					
HOLE Y AUS STACK	IMPLIZIT	7A	4	1	65C02

Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu
ROL Rotate One Bit Left					
VERSCHIEBT NACH LINKS (IM KREIS)	ABS	2E	6	3	
	ZPG	26	5	2	
	ZPG,X	36	6	2	
	ABS,X	3E	6	3	
	AKKU	2A	2	1	
ROR Rotate One Bit Right					
VERSCHIEBT NACH RECHTS (IM KREIS)	ABS	6E	6	3	
	ZPG	66	5	2	
	AKKU	6A	2	1	
	ZPG,X	76	6	2	
	ABS,X	7E	6	3	
RTI Return from Interrupt					
RÜCKSPRUNG AUS INTERRUPT	IMPLIZIT	40	6	1	
RTS Return from Subroutine					
RÜCKSPRUNG AUS SUBROUTINE	IMPLIZIT	60	6	1	

Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu
SBC Substract Memory from Accumulator with Borrow					
SUBTRAHIERT SPEICHER VOM AKKU	IMM	E9	2	2	
	ABS	ED	4	3	
	ZPG	E5	3	2	
	(IND,X)	E1	6	2	
	(IND),Y	F1	5	2	
	ZPG,X	F5	4	2	
	ABS,X	FD	4	3	
	ABS,Y	F9	4	3	
	IND	F2	5	2	65C02
SEC Set Carry Flag					
SETZT CARRY	IMPLIZIT	38	2	1	
SED Set Decimal Mode					
SETZT DEZIMAL	IMPLIZIT	F8	2	1	
SEI Set Interrupt Disable Bit					
SETZT INTERRUPTSPERRE	IMPLIZIT	78	2	1	
Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu

Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu
STA Store Accumulator in Memory					
SPEICHERT	ABS	8D	4	3	
DEN AKKU	ZPG	85	3	2	
	(IND,X)	81	6	2	
	(IND),Y	91	6	2	
	ZPG,X	95	4	2	
	ABS,X	9D	5	3	
	ABS,Y	99	5	3	
	IND	92	5	2	65C02
STX Store Index X in Memory					
SPEICHERT	ABS	8E	4	3	
X REGISTER	ZPG	86	3	2	
	ZPG,Y	96	4	2	
STY Store Index Y in Memory					
SPEICHERT	ABS	8C	4	3	
X REGISTER	ZPG	84	3	2	
	ZPG,Y	94	4	2	
STZ Store Zero in Memory					
SETZE=0	ABS	9C	4	3	65C02
	ZPG	64	3	2	65C02
	ZPG,X	74	4	2	65C02
	ABS,X	9E	5	3	65C02

Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu
TAX Transfer Accumulator to Index X					
TRANSPORTIERT AKKU NACH X	IMPLIZIT	AA	2	1	
TAY Transfer Accumulator to Index Y					
TRANSPORTIERT AKKU NACH Y	IMPLIZIT	A8	2	1	
TRB Test and Reset Memory Bits with Accumulator					
TEST+RESET BITS MIT AKKU	ABS	1C	6	3	65C02
	Z	14	5	2	65C02
TSB Test and Set Memory Bits with Accumulator					
TEST+SET BITS MIT AKKU	ABS	0C	6	3	65C02
	ZPG	04	5	2	65C02
TSX Transfer Stackpointer to Index X					
TRANSPORTIERT STACKZEIGER NACH X	IMPLIZIT	BA	2	1	
TXA Transfer Index X to Accumulator					
TRANSPORTIERT X NACH AKKU	IMPLIZIT	8A	2	1	

Beschreibung	Adress. art	OP Code	Takt zahl	Länge (Byte)	Neu
TXS Transfer Index X to Stackpointer					
TRANSPORTIERT X NACH STACKZEIGER	IMPLIZIT	9A	2	1	
TYA Transfer Index Y to Accumulator					
TRANSPORTIERT Y NACH AKKU	IMPLIZIT	98	2	1	

Nach Gruppen geordnet

MNEMONIC	BESCHREIBUNG	ADRESSART	OP-CODE	
NOP	KEINE OPERATION	IMPLIZIT	EA	
Arithmetische Operationen				
ADC	ADDIERT	IMM	69	
ADC	SPEICHERINHALT	ABS	6D	
ADC	ZUM AKKU	ZPG	65	
ADC	ÄNDERT CARRY	(IND,X)	61	
ADC		(IND),Y	71	
ADC		ZPG,X	75	
ADC		ABS,X	7D	
ADC		ABS,Y	79	
ADC		(ZPG)	72	65C02

MNEMONIC	BESCHREIBUNG	ADRESSART	OP-CODE	
ASL	AKKU ODER SPEICHER	AKKU	0A	
ASL	UM EIN BIT	ABS	0E	
ASL	NACH LINKS	ZPG	06	
ASL	VERSCHIEBEN	ZRP,X	16	
ASL		ABS,X	1E	
CLC	LÖSCHT C	IMPLIZIT	18	
CLD	LÖSCHT DEZIMAL	IMPLIZIT	D8	
CLV	LÖSCHT V	IMPLIZIT	B8	
DEC	REDUZIERT	ABS	CE	
DEC	UM EINS	ZPG	C6	
DEC		ZPG,X	D6	
DEC		ABS,X	DE	
DEA	REDUZIERT AKKU UM 1	AKKU	3A	65C02
DEX	REDUZIERT X UM EINS	IMPLIZIT	CA	
DEY	REDUZIERT Y UM EINS	IMPLIZIT	88	
INC	ADDIERE 1	ABS	EE	
INC		ZPG	E6	
INC		ZPG,X	F6	
INC		ABS,X	FE	
INA	ADD. 1 ZU AKKU	AKKU	1A	65C02
INX	ADD. 1 ZU X	IMPLIZIT	E8	
INY	ADD. 1 ZU Y	IMPLIZIT	C8	

MNEMONIC	BESCHREIBUNG	ADRESSART	OP-CODE	
LSR	AKKU	ABS	4E	
LSR	ODER SPEICHER	ZPG	46	
LSR	UM EIN BIT	ABS	4A	
LSR	NACH RECHTS	ZPG,X	56	
LSR	SCHIEBEN	ABS,X	5E	
ROL	VERSCHIEBT	ABS	2E	
ROL	NACH LINKS	ZPG	26	
ROL	(IM KREIS)	ZPG,X	36	
ROL		ABS,X	3E	
ROL		AKKU	2A	
ROR	VERSCHIEBT	ABS	6E	
ROR	NACH RECHTS	ZPG	66	
ROR	(IM KREIS)	AKKU	6A	
ROR		ZPG,X	76	
ROR		ABS,X	7E	
SBC	SUBTRAHIERT	IMM	E9	
SBC	SPEICHER	ABS	ED	
SBC	VOM AKKU	ZPG	E5	
SBC		(IND,X)	E1	
SBC		(IND),Y	F1	
SBC		ZPG,X	F5	
SBC		ABS,X	FD	
SBC		ABS,Y	F9	
SBC		IND	F2	65C02
SEC	SETZT CARRY	IMPLIZIT	38	
SED	SETZT DEZIMAL	IMPLIZIT	F8	

MNEMONIC	BESCHREIBUNG	ADRESSART	OP-CODE	
Interrupt				
BRK	INTERRUPT	IMPLIZIT	00	
CLI	LÖSCHT INTERRUPTSPERRE	IMPLIZIT	58	
SEI	SETZT INTERRUPTSPERRE	IMPLIZIT	78	
Stackoperationen				
PHA	AKKU ZUM STACK	IMPLIZIT	48	
PHP	PROZESSOR STATUS ZUM STACK	IMPLIZIT	08	
PHX	X REGISTER ZUM STACK	IMPLIZIT	DA	65C02
PHY	Y REGISTER ZUM STACK	IMPLIZIT	5A	65C02
PLA	HOLE AKKU AUS STACK	IMPLIZIT	68	
PLP	HOLE PROZ.STATUS AUS STACK	IMPLIZIT	28	
PLX	HOLE X AUS STACK	IMPLIZIT	FA	65C02
PLY	HOLE Y AUS STACK	IMPLIZIT	7A	65C02

MNEMONIC	BESCHREIBUNG	ADRESSART	OP-CODE	
TSX	TRANSPORTIERT STACKZEIGER NACH X	IMPLIZIT	BA	

Logische Operationen

AND	Log. UND	IMM.	29	
AND	SPEICHER	ABS	2D	
AND	UND AKKU	ZPG	25	
AND		(IND,X)	21	
AND		(IND),Y	31	
AND		ZPG,X	35	
AND		ABS,X	3D	
AND		ABS,Y	39	
AND		(ZPG)	32	65C02
BIT	VERGLEICHT	ABS	2C	
BIT	BIT VON	ZPG	24	
BIT	SPEICHER	IMM	89	
BIT	UND AKKU	ZPG,X	34	65C02
BIT		ABSOL,X	3C	65C02
EOR	LOGISCHES	IMM	49	
EOR	EXKLUSIVES	ABS	4D	
EOR	ODER,	ZPG	45	
EOR	SPEICHER	(IND,X)	41	
EOR	UND AKKU	(IND),Y	51	
EOR		ZPG,X	55	
EOR		ABS,X	5D	
EOR		ABS,Y	59	
EOR		(ZRP)	52	65C02
ORA	LOGISCHES	IMM	09	
ORA	ODER	ABS	0D	
ORA	(SPEICHER	ZPG	05	
ORA	UND AKKU)	(IND,X)	01	

MNEMONIC	BESCHREIBUNG	ADRESSART	OP-CODE	
ORA		(IND),Y	11	
ORA		ZPG,X	15	
ORA		ABS,X	1D	
ORA		ABS,Y	19	
ORA		IND	12	65C02

Speichern, Lesen, Transfer

LDA	LIEST	IMM	A9	
LDA	SPEICHER	ABS	AD	
LDA	IN DEN AKKU	ZPG	A5	
LDA		(IND,X)	A1	
LDA		(IND),Y	B1	
LDA		ZPG,X	B5	
LDA		ABS,X	BD	
LDA		ABS,Y	B9	
LDA		IND	B2	65C02
LDX	LIEST	IMM	A2	
LDX	SPEICHER	ABS	AE	
LDX	IN X REG.	ZPG	A6	
LDX		ZPG,Y	B6	
LDX		ABS,Y	BE	
LDY	LIEST	IMM	A0	
LDY	SPEICHER	ABS	AC	
LDY	IN Y REG.	ZPG	A4	
LDY		ZPG,X	B4	
LDY		ABS,X	BC	
STA	SPEICHERT	ABS	8D	
STA	DEN AKKU	ZPG	85	
STA		(IND,X)	81	
STA		(IND),Y	91	
STA		ZPG,X	95	
STA		ABS,X	9D	

MNEMONIC	BESCHREIBUNG	ADRESSART	OP-CODE	
STA		ABS,Y	99	
STA		IND	92	65C02
STX	SPEICHERT	ABS	8E	
STX	X REGISTER	ZPG	86	
STX		ZPG,Y	96	
STY	SPEICHERT	ABS	8C	
STY	X REGISTER	ZPG	84	
STY		ZPG,Y	94	
STZ	SETZE=0	ABS	9C	65C02
STZ		ZPG	64	65C02
STZ		ZPG,X	74	65C02
STZ		ABS,X	9E	65C02
TAX	TRANSPORTIERT AKKU NACH X	IMPLIZIT	AA	
TAY	TRANSPORTIERT AKKU NACH Y	IMPLIZIT	A8	
TXA	TRANSPORTIERT X NACH AKKU	IMPLIZIT	8A	
TXS	TRANSPORTIERT X NACH STACKZEIGER	IMPLIZIT	9A	
TYA	TRANSPORTIERT	IMPLIZIT	98	

Testbefehle

TRB	TEST+RESET BITS	ABS	1C	65C02
TRB	MIT AKKU	Z	14	65C02

MNEMONIC	BESCHREIBUNG	ADRESSART	OP-CODE	
TSB	TEST+SET BITS	ABS	0C	65C02
TSB	MIT AKKU	ZPG	04	65C02

Verzweigungen

BCC	VERZWEIGUNG BEI CARRY=0	RELATIV	90	
BCS	VERZWEIGUNG BEI CARRY=1	RELATIV	B0	
BEQ	VERZWEIGUNG BEI NULL	RELATIV	F0	
BMI	VERZWEIGT BEI MINUS	RELATIV	30	
BNE	VERZWEIGT BEI >< 0	RELATIV	D0	
BPL	VERZWEIGT BEI PLUS	RELATIV	10	
BRA	UNBED. VERZWEIGUNG	RELATIV	80	65C02
BVC	VERZWEIGT BEI V=0	RELATIV	50	
BVS	VERZWEIGT BEI V=1	REL	70	
CMP	VERGLEICHE	IMM	C9	
CMP	AKKU UND	ZPG	C5	
CMP	SPEICHER	ABS	CD	
CMP		(IND,X)	C1	
CMP		(IND),Y	D1	

MNEMONIC	BESCHREIBUNG	ADRESSART	OP-CODE	
CMP		ZPG,X	D5	
CMP		ABS,X	DD	
CMP		ABS,Y	D9	
CMP		(ZPG)	D2	65C02
CPX	VERGLEICHT	ABS	EC	
CPX	REG. X	IMM	E0	
CPX	UND SPEICHER	ZPG	E4	
CPY	VERGLEICHT	ABS	CC	
CPY	REG. X	IMM	C0	
CPY	UND SPEICHER	ZPG	C4	
JMP	UNBEDINGTER	ABS	4C	
JMP	SPRUNG	IND	6C	
JMP		(IND),Y	7C	65C02
JSR	SPRUNG IN EINE SUBROUTINE	ABS	20	
RTI	RÜCKSPRUNG AUS INTERRUPT	IMPLIZIT	40	
RTS	RÜCKSPRUNG AUS SUBROUTINE	IMPLIZIT	60	

Register

- 103
$ 55
% 55
&GOSUB 164
&GOTO 164
&HOLD 176
&MERGE 176
65C02 CPU 211
80 Zeichen Karte 23, 227
? 42

Abkürzung von Befehlen 128
ABS 96
ACIA 236
Adressen 284
Adressierungsarten 293
Akkumulator 211
Alphabetisches
  Befehlsverzeichnis 242
ALU 211
Alternativer Zeichensatz 87
AND 64
Animation 142
APA 177
APPEND 123
APPLE //e 261
APPLESOFT 41
APPLESOFT Fehlermeldung 265
APPLEWORKS 105, 230
Array 58
ASC 89, 98
ASCII Code 89, 277
ASSEMBLER 201, 210
Assemblerbefehle 298, 313
Assemblerprogramme 207
Assemblersyntax 211
ATN 97, 242
Ausgabeprogramm 91

Backslash 43
Bank 263
Bankswitching 105
BASIC 41
BASIC Compiler 169
BASIC, strukturiert 162
BASIC.SYSTEM 38
Baudrate 230
Befehlssatz CPU 298
Befehlsverzeichnis 242
Bell 58
Betriebssystem 22, 98
Betriebssystembefehle 257
Betriebssysteme 29
Bewegte Bilder 142
Bildschirm 18
Binaerfile 206
BLOAD 242
Blockweise Speicherung 126
Bool'sche Operation 64
Booten 37
BREAK 172
BRUN 242
BSAVE 243
Byteweiser Filezugriff 128

CALL 108
CAPS LOCK 17
Carriage Return 233
Carrybit 218
CAT 27, 46
CATALOG 27
CHAIN 101, 160, 183
Chaining 159
CHR$ 89, 98
CHR$(4) 99, 111
CHR$(13) 100
CLEAR 243
CLOSE 111, 115
COBOL 187
Codefile 189
COLOR= 130
Compiler 169, 186, 194
CONT 172
CONTROL-A 231
CONTROL-B 225
CONTROL-C 224
CONTROL-D 99, 111

CONTROL-G 58
CONTROL-I 231
CONTROL-RESET 224
CONTROL-S 174
CONTROL-V 235
CONTROL-W 235
COS 97
CP/M 22
CPU 211, 260
CR 233
CREATE 101

DATA 92
Datenformat 231
Datum 31
DEZIMAL 199
DEF FN 97
DEL 53
DELETE 244
Dialekte 41, 201
DIM 59
Dimensionierung 59
Directory 32
Disassembler 204, 221
DO..LOOP 198
Doppelseitige Disketten 39
DOS3.3 34
DOS Fehlermeldungen 266
DRAW 138

Echo 233
Editor 50, 189, 200
Eingabeprogramm 90
Einsprungadressen 284
Einzelanweisung 42
END 173
END OF DATA 122
Errorhandling 171
Errormessages 265
esc 50
ESC-4, ESC-8 88
Escape-Editor 50
EXEC 245
EXP 96
Externes Laufwerk 20, 227

Farben 132
Fehlerabfangung 171, 175
Fehlercode 269
Fehlermeldungen 265
Fehlersuche 171
Feld 58, 118
Feldweiser Filezugriff 127
Fernseher 18, 226
Festkommavariable 55
File 28
File(s) still open 122
Filename 49
Filepuffer 275
Filestruktur 31
Filetypen 105
Filetypen (ProDOS) 129
Filezugriff (ProDOS) 128
Filezugriff, Kontrolle 174
FLASH 88
FLUSH 128
FN 97
FOR..NEXT 73
Formatieren 35
FORTH 191
FORTH, Editor 200
FORTH, Screen 201
FORTH, Zahlensystem 199
FORTRAN 187
FP 101
FRE 60, 167
FRE(0) 56, 167
Funktionen 95

Garbage Collection 167
Genauigkeit 57
GET 81, 122
Gleitkommavariable 55
GOSUB 69
GOTO 67
GR 130
Grafik 130
Grafik invertieren 152
Grafik speichern 144, 151
Grafik umschalten 132, 136
Grafik, hochauflösend 133
Grafik, niedrigauflösend 130
Grafik, Speicherbedarf 141

Hardcopy 159
Hardwarepage 264, 293
HCOLOR= 135
HELLO 34
Hexadezimalsystem 110
HGR 133
HGR2 134
Hilfsprogramme 176
Hilfsroutinen 173, 228
HIMEM: 142, 247
HLIN 131
HOME 50
HPLOT 135
HTAB 85

F..THEN 77
IN# 227
Inhaltsverszeichnis 27
INIT 36, 101
INPUT 79, 113
INT 248
INT (Applesoft) 96
Integer BASIC 42, 208
Integervariable 55
Interpreter 186, 194
Interrupt 218
INVERSE 87
Invertieren, Grafik 152

Kompakte Programme 166
Konfigurieren der Ports 229
Kontrollzeichen 118, 235, 277

Laufwerk 20
LEFT$ 98
LEN 98
LET 248
Linefeed 234
Linienfolgen 136
LIST 44
Listing, Masch.programm 205
LOAD 46
LOCK 48
LOG 96
LOGO 187
LOMEM: 142, 249

Magnetfeld 18
Maschinenadressen 284
Maschinenbefehle 299
Maschinenbefehle, Typen 213
Maschinenprogramme 108, 201 ff., 284
Matrizen 58
Maus 146
Maus, auschalten 147
Maus, Aktivierung 146
Maus, Anschluss 283
Maus, Firmware 152
Maus, Grafik 148
Maus, Routinen 152
Mauszeichen 88, 280
MAXFILES 101, 115, 249
mehrdimensionale Arrays 59
MID$ 98, 249
Microsoft 42
Mikroprozessor 212
Miniassembler 208
Mnemonics 207, 299
MODULA 2 188
MON C,I,O 102, 174
Monitor 219, 263
MOUSEPAINT 145, 148
MS-DOS 22

NEW 49
NEXT 73, 250
NOMON 102, 174
NORMAL 87
NOT 64
Notation, umgekehrte 195
NOTRACE 172
Nummerierung Speicher 106

OK 193
ON..GOSUB 71, 250
ON..GOTO 68, 250
ONERR 123, 175, 250
OPEN 111 ,113, 251
OR 64

Parameter von Befehlen 104
Parity 231
PASCAL 188
PDL 251
PEEK 106
Pinbelegung 281
PLOT 131
POKE 107
Polnische Notation 195
POP 72, 251
Port 226
POSITION 124
PR# 227
PR# 251
PR#0, PR#3 88
PREFIX 33, 102
PRINT 42, 82, 252
Procedure 189
ProDOS 30
ProDOS Fehlermeldungen 267
Programcounter 211
Programm aendern 178
Programm, kompakt 166
Programm, schnelles 167
Programm, verzoegern 174
Programmentwicklung 170
Programmierhilfen 169
Programmschleifen 74
Programmiersprachen 186
Programmverkettung 159
Prozedur 189
Prozessor 212
Prozessorinhalt 223

RAM+Disk 181
RAM Disk, Daten 184
RAM Disk, Grafik 185
RAM Disk, Programme 182
RANDOM ACCESS File 117, 124
Rangordnung Operationen 61
READ (Applesoft) 92, 252
READ (DOS/ProDOS) 111, 252
Record 118
Register 211
REM 253
RENAME 49
RENUMBER 156, 176
Reservierte Worte 271
RESTORE (Applesoft) 94, 253
RESTORE (ProDOS) 103, 160, 253
RESUME 175, 253
RETURN 69
RIGHT$ 98
RND 96
ROT 138
RUN 44

Schachtelung 70
SAVE 46
SCALE 138
Schleifen 73
Schnelle Programme 167
SCRN 132
Sektor 20, 35
Sequentielles File 117
Serielle Ports/Slots 229, 281
SGN 96
Shapes 136
SIN 96
Slot 226
SPC( ) 84
SPEED= 88, 174
Speicheradressen 284
Speicheraufteilung 109, 263, 273
Speicherbedarf 60
Speicherinhalte 219
Speicherplatz, freier 60
Speichernummerierung 106
Spur 20
SQR 96
Stack 196
Stackpointer 211
Stapelspeicher 196
STARTUP 31
STEP 75
STOP 172
STORE 103, 160
Stringvariable 55
Stromversorgung 20, 281
Strukturiertes BASIC 162
Strukturierung 155
Subdirectory 32
Subroutine 155
Syntax, Assembler 211
Systemprogramme 263

TAB 85
Taktfrequenz 262
TAN 97
Tastatur 16
Terminalmodus 235
TEXT 132
Textfile 112
Textverarbeitung 178
Textvergleiche 64
THEN 77
Tool Kit 177
TRACE 171
Track 20
TURBO-Pascal 188

UCSD Pascal 34, 188
Uebergabeparameter 155
Uhrzeit 31
UNLOCK 49
Unterprogramme 156
Unterschiede Apple //e,c 261
USR( ) 255

VAL 80, 98
Variable 54
Vergleiche 63
VERIFY 47
Verzweigung 66
Videoausgang 282
VLIN 132
Vorspann 139
VTAB 85

WAIT 255
Window 88
Wort 192
WRITE 111, 255

XDRAW 140

Z 80 Prozessor 23, 188
ZAP 234
Zeichensatz 87
Zeilenbreite 233
Zeilennummer 44
Zugriffsgeschwindigkeit 181

Das Buch wendet sich an Apple//c Besitzer, die die Leistungsfähigkeit ihres Computers bisher noch nicht ausschöpfen konnten.

Leicht verständlich, trotzdem ausführlich wird die Sprache APPLESOFT Basic dargestellt. Eine Vielzahl von Programmen, die speziell auf den Apple//c zugeschnitten wurden, zeigen die Wirkung der einzelnen Befehle bis zu Programmiertechniken für Diskettenzugriff, Maus und andere Anwendungen.

Apple//c Maschinensprache, Programmeingabe und -kontrolle mit Monitorbefehlen werden ausführlich dargestellt. Nützliche Utilities zur Datumseingabe oder zum direkten Zugriff auf Directories eröffnen zusätzliche Möglichkeiten.

Kurze Einführungen in PASCAL und FORTH geben dem Leser eine Basis für die Programmierung des Apple//c in anderen Sprachen.

Die Konfigurierung der seriellen Ports und weitere spezifische Beschreibungen des Computers ermöglichen die gezielte Kontrolle über die Maschine.

Mehr als 90 PEEK-, POKE- und Maschinenprogrammadressen, Speicheraufteilung und weitere Tabellen sind für den Programmierer eine konzentrierte, wichtige Grundlage.